**Date:** May 2016

Semantic Information Modeling for Federation (*SIMF*)

*Version #0.8*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**OMG Document Number:** 2016-05-02

Manifest  ad/2016-05-01  
Submission document ad/2016-05-02 (this document)  
SIMF Conceptual model XMI ad/2016-05-03  
SIMF MOF Meta model XMI ad/2016-05-04  
SIMF Profile XMI ad/2016-05-05  
SIMF Profile mapping XMI ad/2016-05-06  
SIMF OWL Mapping XMI ad/2016-06-07  
SIMF Kernel Formalization ad/2016-05-08  
Non-normative artifacts (ZIP) ad/2016-05-09

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Copyright © 2016, Object Management Group, Inc.

Copyright © 2016, Data Access Technologies, Inc. (Model Driven Solutions Division)

Copyright © 2016, PNA-Group, Ltd.

Copyright © 2016, No Magic Inc.

Copyright © 2016, 88 Solutions, Inc.

Copyright © 2016, Thematix Partners LLC

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 109 Highland Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS

IMM®, MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™ , Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™ , MOF™ , OMG Interface Definition Language (IDL)™ , and OMG SysML™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

**OMG’s Issue Reporting Procedure**

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents, Report a Bug/Issue (http://www.omg.org/report\_issue.)

Table of Contents

[1 Scope 13](#_Toc450571067)

[1.1 Business Need 13](#_Toc450571068)

[1.2 Approach 16](#_Toc450571069)

[1.3 Unified Meta Model & Notation 17](#_Toc450571070)

[2 Conformance 17](#_Toc450571071)

[3 References 18](#_Toc450571072)

[4 Terms and Definitions 19](#_Toc450571073)

[5 Symbols 19](#_Toc450571074)

[6 Additional Information 19](#_Toc450571075)

[6.1 Changes to Adopted OMG Specifications [optional] 19](#_Toc450571076)

[6.2 Acknowledgments 19](#_Toc450571077)

[2 Introduction to SIMF Concepts 21](#_Toc450571078)

[*2.1* *Pragmatic world view* 21](#_Toc450571079)

[*2.2* *Models* 22](#_Toc450571080)

[*2.3* *Concepts* 23](#_Toc450571081)

[2.3.1 Dictionary Concepts 24](#_Toc450571082)

[*2.4* *Information, Values ands Anything* 25](#_Toc450571083)

[*2.5* *Context* 25](#_Toc450571084)

[*2.6* *Entities* 26](#_Toc450571085)

[*2.7* *Situations* 26](#_Toc450571086)

[2.8 Relations 26](#_Toc450571087)

[*2.9* *Types* 26](#_Toc450571088)

[2.9.1 Individual Types 26](#_Toc450571089)

[2.9.2 Value types 26](#_Toc450571090)

[2.9.3 Situation Types & Situational Roles 27](#_Toc450571091)

[2.9.4 Relation Types 27](#_Toc450571092)

[2.10 Rules 27](#_Toc450571093)

[2.10.1 Production Rules 27](#_Toc450571094)

[2.10.2 As-A Rule 27](#_Toc450571095)

[2.10.3 Logical Grounding Rules 28](#_Toc450571096)

[2.10.4 Constraints 28](#_Toc450571097)

[2.10.5 General Constraints 28](#_Toc450571098)

[2.10.6 Multiplicity Constraints 28](#_Toc450571099)

[2.10.7 Uniqueness Constraints 28](#_Toc450571100)

[2.11 Quantifiers 28](#_Toc450571101)

[2.12 Expressions 28](#_Toc450571102)

[2.13 Properties & Units 29](#_Toc450571103)

[2.14 Naming 29](#_Toc450571104)

[2.15 Representations 29](#_Toc450571105)

[2.16 Important Relations 29](#_Toc450571106)

[2.16.1 Contextualization 29](#_Toc450571107)

[2.16.2 Equivalence 29](#_Toc450571108)

[2.16.3 Generalization 30](#_Toc450571109)

[2.17 SIMF Lexical Scope & Physical Representations 30](#_Toc450571110)

[2.17.1 Transactions 30](#_Toc450571111)

[3 Abstract Syntax [Normative] 32](#_Toc450571112)

[3.1 SIMF Conceptual Model 33](#_Toc450571113)

[3.1.1 Diagram: SIMF Packages 33](#_Toc450571114)

[3.2 SIMF Conceptual Model::Expressions 34](#_Toc450571115)

[3.2.1 Diagram: Expressions 34](#_Toc450571116)

[3.2.2 Class Computed Fact 34](#_Toc450571117)

[3.2.3 Class Constant Reference 34](#_Toc450571118)

[3.2.31 Direct Supertypes 34](#_Toc450571119)

[3.2.32 Associations 35](#_Toc450571120)

[3.2.4 Association Constant Value 35](#_Toc450571121)

[3.2.41 Association Ends 35](#_Toc450571122)

[3.2.5 Class Equality 35](#_Toc450571123)

[3.2.51 Direct Supertypes 35](#_Toc450571124)

[3.2.52 Associations 35](#_Toc450571125)

[3.2.6 Class Evaluation 35](#_Toc450571126)

[3.2.61 Direct Supertypes 35](#_Toc450571127)

[3.2.62 Associations 35](#_Toc450571128)

[3.2.7 Class Expression 35](#_Toc450571129)

[3.2.71 Direct Supertypes 35](#_Toc450571130)

[3.2.72 Attributes 36](#_Toc450571131)

[3.2.73 Associations 36](#_Toc450571132)

[3.2.8 Association Expression Context 36](#_Toc450571133)

[3.2.81 Association Ends 36](#_Toc450571134)

[3.2.9 Class Expression Context 36](#_Toc450571135)

[3.2.91 Direct Supertypes 36](#_Toc450571136)

[3.2.92 Associations 36](#_Toc450571137)

[3.2.10 Class Function Call 36](#_Toc450571138)

[3.2.101 Direct Supertypes 36](#_Toc450571139)

[3.2.102 Associations 36](#_Toc450571140)

[3.2.11 Association Function Implementation 37](#_Toc450571141)

[3.2.111 Association Ends 37](#_Toc450571142)

[3.2.12 Class Function Type 37](#_Toc450571143)

[3.2.121 Direct Supertypes 37](#_Toc450571144)

[3.2.122 Associations 37](#_Toc450571145)

[3.2.13 Association Function Type 37](#_Toc450571146)

[3.2.131 Association Ends 37](#_Toc450571147)

[3.2.14 Class Object Operation Type 37](#_Toc450571148)

[3.2.141 Direct Supertypes 37](#_Toc450571149)

[3.2.142 Associations 37](#_Toc450571150)

[3.2.15 Association OO Target 38](#_Toc450571151)

[3.2.151 Association Ends 38](#_Toc450571152)

[3.2.16 Association Return type 38](#_Toc450571153)

[3.2.161 Association Ends 38](#_Toc450571154)

[3.2.17 Class Traversal 38](#_Toc450571155)

[3.2.171 Direct Supertypes 38](#_Toc450571156)

[3.2.172 Attributes 38](#_Toc450571157)

[3.2.173 Associations 38](#_Toc450571158)

[3.2.18 Association traverse through 39](#_Toc450571159)

[3.2.181 Association Ends 39](#_Toc450571160)

[3.3 SIMF Conceptual Model::Identifiers 40](#_Toc450571161)

[3.3.1 Diagram: Identifiers 40](#_Toc450571162)

[3.3.2 Association Identification 40](#_Toc450571163)

[3.3.21 Association Ends 40](#_Toc450571164)

[3.3.3 Class Identifier 40](#_Toc450571165)

[3.3.31 Direct Supertypes 40](#_Toc450571166)

[3.3.32 Associations 40](#_Toc450571167)

[3.3.4 Class IRI Identifier 41](#_Toc450571168)

[3.3.41 Direct Supertypes 41](#_Toc450571169)

[3.3.5 Class Term 41](#_Toc450571170)

[3.3.51 Direct Supertypes 41](#_Toc450571171)

[3.3.6 Class Text Identifier 41](#_Toc450571172)

[3.3.61 Direct Supertypes 41](#_Toc450571173)

[3.4 SIMF Conceptual Model::Kernel 42](#_Toc450571174)

[3.4.1 Diagram: SIMF Kernel Base 42](#_Toc450571175)

[3.4.2 Diagram: SIMF Kernel Rules 43](#_Toc450571176)

[3.4.3 Diagram: SIMF Kernel Values 44](#_Toc450571177)

[3.5 SIMF Conceptual Model::Lexical Scope 45](#_Toc450571178)

[3.5.1 Diagram: Lexical scope 45](#_Toc450571179)

[3.5.2 Association Definition 45](#_Toc450571180)

[3.5.21 Association Ends 46](#_Toc450571181)

[3.5.3 Class Definition 46](#_Toc450571182)

[3.5.31 Direct Supertypes 46](#_Toc450571183)

[3.5.32 Attributes 46](#_Toc450571184)

[3.5.33 Associations 46](#_Toc450571185)

[3.5.4 Association Documentation 46](#_Toc450571186)

[3.5.41 Association Ends 46](#_Toc450571187)

[3.5.5 Class Include 46](#_Toc450571188)

[3.5.51 Direct Supertypes 46](#_Toc450571189)

[3.5.6 Class Lexical Reference 47](#_Toc450571190)

[3.5.61 Direct Supertypes 47](#_Toc450571191)

[3.5.62 Associations 47](#_Toc450571192)

[3.5.7 Class Lexical Scope 47](#_Toc450571193)

[3.5.71 Direct Supertypes 47](#_Toc450571194)

[3.5.72 Associations 47](#_Toc450571195)

[3.5.8 Class Model 47](#_Toc450571196)

[3.5.81 Direct Supertypes 47](#_Toc450571197)

[3.5.82 Associations 47](#_Toc450571198)

[3.5.9 Class Package 47](#_Toc450571199)

[3.5.91 Direct Supertypes 47](#_Toc450571200)

[3.5.92 Attributes 47](#_Toc450571201)

[3.5.10 Association Reference 48](#_Toc450571202)

[3.5.101 Association Ends 48](#_Toc450571203)

[3.5.11 Association Scope 48](#_Toc450571204)

[3.5.111 Association Ends 48](#_Toc450571205)

[3.5.12 Class term 48](#_Toc450571206)

[3.5.121 Enumeration ModelKind 48](#_Toc450571207)

[3.5.121 Literals 48](#_Toc450571208)

[3.6 SIMF Conceptual Model::Mapping 49](#_Toc450571209)

[3.6.1 Diagram: Facades 49](#_Toc450571210)

[3.6.2 Diagram: Mapping Rules 50](#_Toc450571211)

[3.6.3 Class Computed Facade 50](#_Toc450571212)

[3.6.31 Direct Supertypes 50](#_Toc450571213)

[3.6.32 Operations 50](#_Toc450571214)

[3.6.4 Class Facade 51](#_Toc450571215)

[3.6.41 Direct Supertypes 51](#_Toc450571216)

[3.6.5 Association from rue 51](#_Toc450571217)

[3.6.51 Association Ends 51](#_Toc450571218)

[3.6.6 Class Map Rule 51](#_Toc450571219)

[3.6.61 Direct Supertypes 51](#_Toc450571220)

[3.6.62 Associations 51](#_Toc450571221)

[3.6.7 Class Map Rule End 51](#_Toc450571222)

[3.6.71 Attributes 51](#_Toc450571223)

[3.6.72 Associations 51](#_Toc450571224)

[3.6.8 Association Map Rule Type Assertion 52](#_Toc450571225)

[3.6.81 Association Ends 52](#_Toc450571226)

[3.6.9 Association Mapped Property 52](#_Toc450571227)

[3.6.91 Association Ends 52](#_Toc450571228)

[3.6.10 Association Mapped Relationship 52](#_Toc450571229)

[3.6.101 Association Ends 52](#_Toc450571230)

[3.6.11 Class Mapping 52](#_Toc450571231)

[3.6.111 Direct Supertypes 52](#_Toc450571232)

[3.6.112 Associations 52](#_Toc450571233)

[3.6.12 Association pattern for 52](#_Toc450571234)

[3.6.121 Association Ends 53](#_Toc450571235)

[3.6.13 Class Property End 53](#_Toc450571236)

[3.6.131 Direct Supertypes 53](#_Toc450571237)

[3.6.132 Associations 53](#_Toc450571238)

[3.6.14 Class Relationship End 53](#_Toc450571239)

[3.6.141 Direct Supertypes 53](#_Toc450571240)

[3.6.142 Associations 53](#_Toc450571241)

[3.6.15 Association to rule 53](#_Toc450571242)

[3.6.151 Association Ends 53](#_Toc450571243)

[3.7 SIMF Conceptual Model::Patterns 54](#_Toc450571244)

[3.7.1 Diagram: Patterns 54](#_Toc450571245)

[3.7.2 Class Pattern 54](#_Toc450571246)

[3.7.21 Direct Supertypes 54](#_Toc450571247)

[3.7.22 Associations 55](#_Toc450571248)

[3.7.3 Class Pattern property 55](#_Toc450571249)

[3.7.31 Direct Supertypes 55](#_Toc450571250)

[3.7.32 Attributes 55](#_Toc450571251)

[3.7.33 Associations 55](#_Toc450571252)

[3.7.34 Enumeration Quantifier 55](#_Toc450571253)

[3.7.34 Literals 55](#_Toc450571254)

[3.8 SIMF Conceptual Model::Records 57](#_Toc450571255)

[3.8.1 Diagram: Records 57](#_Toc450571256)

[3.8.2 Class Record 57](#_Toc450571257)

[3.8.21 Direct Supertypes 57](#_Toc450571258)

[3.8.22 Associations 57](#_Toc450571259)

[3.8.3 Class Record Type 57](#_Toc450571260)

[3.8.31 Direct Supertypes 58](#_Toc450571261)

[3.8.32 Associations 58](#_Toc450571262)

[3.9 SIMF Conceptual Model::Relationships 59](#_Toc450571263)

[3.9.1 Diagram: Annotations 59](#_Toc450571264)

[3.9.2 Diagram: Relationships 60](#_Toc450571265)

[3.9.3 Class Annotation Property 60](#_Toc450571266)

[3.9.31 Direct Supertypes 60](#_Toc450571267)

[3.9.32 Associations 60](#_Toc450571268)

[3.9.4 Class Annotation Relationship Type 61](#_Toc450571269)

[3.9.41 Direct Supertypes 61](#_Toc450571270)

[3.9.42 Associations 61](#_Toc450571271)

[3.9.5 Class Relationship 61](#_Toc450571272)

[3.9.51 Direct Supertypes 61](#_Toc450571273)

[3.9.52 Associations 61](#_Toc450571274)

[3.9.6 Class Relationship Type 61](#_Toc450571275)

[3.9.61 Direct Supertypes 61](#_Toc450571276)

[3.9.62 Associations 61](#_Toc450571277)

[3.10 SIMF Conceptual Model::Rules 62](#_Toc450571278)

[3.10.1 Diagram: Rules 62](#_Toc450571279)

[3.10.2 Class Conditional Constraint 62](#_Toc450571280)

[3.10.21 Direct Supertypes 62](#_Toc450571281)

[3.10.22 Attributes 63](#_Toc450571282)

[3.10.3 Class Covering Constraint 63](#_Toc450571283)

[3.10.31 Direct Supertypes 63](#_Toc450571284)

[3.10.32 Associations 63](#_Toc450571285)

[3.10.4 Class Disjoint 63](#_Toc450571286)

[3.10.41 Direct Supertypes 63](#_Toc450571287)

[3.10.5 Class Enumerated 63](#_Toc450571288)

[3.10.51 Direct Supertypes 63](#_Toc450571289)

[3.10.6 Class Equivalent 63](#_Toc450571290)

[3.10.61 Direct Supertypes 63](#_Toc450571291)

[3.10.7 Association Generalizations 64](#_Toc450571292)

[3.10.71 Association Ends 64](#_Toc450571293)

[3.10.8 Class Multiplicity Constraint 64](#_Toc450571294)

[3.10.81 Direct Supertypes 64](#_Toc450571295)

[3.10.82 Attributes 64](#_Toc450571296)

[3.10.83 Associations 64](#_Toc450571297)

[3.10.9 Association Multiplicity Perspective 65](#_Toc450571298)

[3.10.91 Association Ends 65](#_Toc450571299)

[3.10.10 Association Multiplicity Target 65](#_Toc450571300)

[3.10.101 Association Ends 65](#_Toc450571301)

[3.10.11 Class Pattern Constraint 65](#_Toc450571302)

[3.10.111 Direct Supertypes 65](#_Toc450571303)

[3.10.12 Class Property Constraint 65](#_Toc450571304)

[3.10.121 Direct Supertypes 65](#_Toc450571305)

[3.10.122 Associations 65](#_Toc450571306)

[3.10.13 Class Property Generalization Constraint 65](#_Toc450571307)

[3.10.131 Direct Supertypes 66](#_Toc450571308)

[3.10.132 Attributes 66](#_Toc450571309)

[3.10.133 Associations 66](#_Toc450571310)

[3.10.14 Association Property Generalizations 66](#_Toc450571311)

[3.10.141 Association Ends 66](#_Toc450571312)

[3.10.15 Association Property Specializations 66](#_Toc450571313)

[3.10.151 Association Ends 66](#_Toc450571314)

[3.10.16 Class Property Transitivity Constraint 66](#_Toc450571315)

[3.10.161 Direct Supertypes 67](#_Toc450571316)

[3.10.17 Association Property Type 67](#_Toc450571317)

[3.10.171 Association Ends 67](#_Toc450571318)

[3.10.18 Class Property Type Constraint 67](#_Toc450571319)

[3.10.181 Direct Supertypes 67](#_Toc450571320)

[3.10.182 Attributes 67](#_Toc450571321)

[3.10.183 Associations 67](#_Toc450571322)

[3.10.19 Class Rule 67](#_Toc450571323)

[3.10.191 Direct Supertypes 67](#_Toc450571324)

[3.10.192 Associations 67](#_Toc450571325)

[3.10.20 Association Rule constrains 68](#_Toc450571326)

[3.10.201 Association Ends 68](#_Toc450571327)

[3.10.21 Association Specializations 68](#_Toc450571328)

[3.10.211 Association Ends 68](#_Toc450571329)

[3.10.22 Class Type Constraint 68](#_Toc450571330)

[3.10.221 Direct Supertypes 68](#_Toc450571331)

[3.10.222 Associations 68](#_Toc450571332)

[3.10.23 Class Type Generalization Constraint 68](#_Toc450571333)

[3.10.231 Direct Supertypes 68](#_Toc450571334)

[3.10.232 Attributes 68](#_Toc450571335)

[3.10.233 Associations 69](#_Toc450571336)

[3.10.24 Class Type or Property 69](#_Toc450571337)

[3.10.241 Associations 69](#_Toc450571338)

[3.10.25 Association Unique Set 69](#_Toc450571339)

[3.10.251 Association Ends 69](#_Toc450571340)

[3.10.26 Class Uniqueness Constraint 69](#_Toc450571341)

[3.10.261 Direct Supertypes 69](#_Toc450571342)

[3.10.262 Attributes 69](#_Toc450571343)

[3.10.263 Associations 69](#_Toc450571344)

[3.11 SIMF Conceptual Model::Situations 70](#_Toc450571345)

[3.11.1 Diagram: Situations 70](#_Toc450571346)

[3.11.2 Class Binding 70](#_Toc450571347)

[3.11.21 Direct Supertypes 71](#_Toc450571348)

[3.11.22 Associations 71](#_Toc450571349)

[3.11.3 Association bound in situation 71](#_Toc450571350)

[3.11.31 Association Ends 71](#_Toc450571351)

[3.11.4 Association bound individual 71](#_Toc450571352)

[3.11.41 Association Ends 71](#_Toc450571353)

[3.11.5 Association bound property 71](#_Toc450571354)

[3.11.51 Association Ends 71](#_Toc450571355)

[3.11.6 Association Properties 71](#_Toc450571356)

[3.11.61 Association Ends 72](#_Toc450571357)

[3.11.7 Class Property 72](#_Toc450571358)

[3.11.71 Direct Supertypes 72](#_Toc450571359)

[3.11.72 Associations 72](#_Toc450571360)

[3.11.8 Class Situation 72](#_Toc450571361)

[3.11.81 Direct Supertypes 72](#_Toc450571362)

[3.11.82 Associations 72](#_Toc450571363)

[3.11.9 Class Situation Type 73](#_Toc450571364)

[3.11.91 Direct Supertypes 73](#_Toc450571365)

[3.11.92 Associations 73](#_Toc450571366)

[3.11.10 Class Structure 73](#_Toc450571367)

[3.11.101 Direct Supertypes 73](#_Toc450571368)

[3.11.102 Associations 73](#_Toc450571369)

[3.11.11 Class Structured Type 73](#_Toc450571370)

[3.11.111 Direct Supertypes 73](#_Toc450571371)

[3.11.112 Associations 73](#_Toc450571372)

[3.12 SIMF Conceptual Model::Top level 74](#_Toc450571373)

[3.12.1 Diagram: Context 74](#_Toc450571374)

[3.12.2 Diagram: TopLevel 74](#_Toc450571375)

[3.12.3 Class Anything 75](#_Toc450571376)

[3.12.31 Associations 75](#_Toc450571377)

[3.12.4 Association Assertion 75](#_Toc450571378)

[3.12.41 Association Ends 75](#_Toc450571379)

[3.12.5 Class Context 75](#_Toc450571380)

[3.12.51 Direct Supertypes 75](#_Toc450571381)

[3.12.52 Associations 75](#_Toc450571382)

[3.12.6 Class Entity 75](#_Toc450571383)

[3.12.61 Direct Supertypes 75](#_Toc450571384)

[3.12.62 Associations 76](#_Toc450571385)

[3.12.7 Association In Context 76](#_Toc450571386)

[3.12.71 Association Ends 76](#_Toc450571387)

[3.12.8 Class Proposition 76](#_Toc450571388)

[3.12.81 Direct Supertypes 76](#_Toc450571389)

[3.12.82 Associations 76](#_Toc450571390)

[3.12.9 Association Term Preference 76](#_Toc450571391)

[3.12.91 Association Ends 76](#_Toc450571392)

[3.13 SIMF Conceptual Model::Types 77](#_Toc450571393)

[3.13.1 Diagram: Type-instance 77](#_Toc450571394)

[3.13.2 Diagram: Types 78](#_Toc450571395)

[3.13.3 Class Facet 78](#_Toc450571396)

[3.13.31 Direct Supertypes 78](#_Toc450571397)

[3.13.4 Association Instance Relation 79](#_Toc450571398)

[3.13.5 Direct Supertypes 79](#_Toc450571399)

[3.13.51 Association Ends 79](#_Toc450571400)

[3.13.6 Class Intersection Type 79](#_Toc450571401)

[3.13.61 Direct Supertypes 79](#_Toc450571402)

[3.13.7 Class Phase 79](#_Toc450571403)

[3.13.71 Direct Supertypes 79](#_Toc450571404)

[3.13.8 Class Role 79](#_Toc450571405)

[3.13.81 Direct Supertypes 79](#_Toc450571406)

[3.13.9 Class Type 79](#_Toc450571407)

[3.13.91 Direct Supertypes 79](#_Toc450571408)

[3.13.92 Associations 80](#_Toc450571409)

[3.13.10 Class Union Type 80](#_Toc450571410)

[3.13.101 Direct Supertypes 80](#_Toc450571411)

[3.14 SIMF Conceptual Model::Values 81](#_Toc450571412)

[3.14.1 Diagram: Values 82](#_Toc450571413)

[3.14.2 Class Base Unit Type 82](#_Toc450571414)

[3.14.21 Direct Supertypes 83](#_Toc450571415)

[3.14.3 Class Boolean 83](#_Toc450571416)

[3.14.31 Direct Supertypes 83](#_Toc450571417)

[3.14.4 Class Number 83](#_Toc450571418)

[3.14.41 Direct Supertypes 83](#_Toc450571419)

[3.14.5 Class Quantity kind 83](#_Toc450571420)

[3.14.51 Direct Supertypes 83](#_Toc450571421)

[3.14.6 Class Quantity Value 83](#_Toc450571422)

[3.14.61 Direct Supertypes 83](#_Toc450571423)

[3.14.62 Associations 84](#_Toc450571424)

[3.14.7 Class Text 84](#_Toc450571425)

[3.14.71 Direct Supertypes 84](#_Toc450571426)

[3.14.8 Class Unit Type 84](#_Toc450571427)

[3.14.81 Direct Supertypes 84](#_Toc450571428)

[3.14.82 Attributes 84](#_Toc450571429)

[3.14.83 Associations 85](#_Toc450571430)

[3.14.9 Class Value 85](#_Toc450571431)

[3.14.91 Direct Supertypes 85](#_Toc450571432)

[3.14.10 Class Value Type 85](#_Toc450571433)

[3.14.101 Direct Supertypes 85](#_Toc450571434)

[3.14.102 Associations 85](#_Toc450571435)

[4 Foundational Assumptions (Normative) 85](#_Toc450571436)

[4.1 Multiple representations of overlapping concepts 85](#_Toc450571437)

[4.2 Models include data 85](#_Toc450571438)

[4.3 Conceptual Models 85](#_Toc450571439)

[4.4 Identity and identifiers 86](#_Toc450571440)

[4.5 Facts 86](#_Toc450571441)

[4.6 Representations of a CONCEPT 86](#_Toc450571442)

[4.6.1 Relation 86](#_Toc450571443)

[4.7 Representation identifiers 87](#_Toc450571444)

[4.7.1 Example physical identifiers 87](#_Toc450571445)

[4.8 CONCEPT representation identifiers 87](#_Toc450571446)

[4.9 Sources 87](#_Toc450571447)

[4.9.1 Relation: 87](#_Toc450571448)

[4.10 Ownership 88](#_Toc450571449)

[4.11 Lifetime and context of facts 88](#_Toc450571450)

[5 Annex A Examples (informative) 90](#_Toc450571451)

[6 Mapping to OWL 2 (normative) 91](#_Toc450571452)

[6.1 Class 91](#_Toc450571453)

[6.2 Class Generalization 91](#_Toc450571454)

[6.3 Class with Datatype Property 92](#_Toc450571455)

[6.4 Class with Self-Referential Object Property 92](#_Toc450571456)

[6.5 Class with Object Property 93](#_Toc450571457)

[6.6 Property Holder with Datatype Property 93](#_Toc450571458)

[6.7 Property Holder with Self-Referential Object Property 94](#_Toc450571459)

[6.8 Property Holder with Object Property 94](#_Toc450571460)

[6.9 Class with Object Property without Range 95](#_Toc450571461)

[6.10 Class with Subproperty 95](#_Toc450571462)

[6.11 Class with Universal Quantification Constraint on Property I 96](#_Toc450571463)

[6.12 Class with Universal Quantification Constraint on Property II 97](#_Toc450571464)

[6.13 Class with Existential Quantification Constraint on Property 98](#_Toc450571465)

[6.14 Property Holder with Self-Referential Subproperty 98](#_Toc450571466)

[6.15 Property Holder with Subproperty 99](#_Toc450571467)

[6.16 Class with Subproperty without a Range 100](#_Toc450571468)

[6.17 Class with Necessary and Sufficient Property 101](#_Toc450571469)

[6.18 Class With Property Having Unspecified Multiplicity 102](#_Toc450571470)

[7 Annex A: UML Conceptual Modeling Profile (normative) 103](#_Toc450571471)

[7.1 UML Conceptual Model Primer 103](#_Toc450571472)

[7.1.11 Classes 103](#_Toc450571473)

[7.1.12 Instances 104](#_Toc450571474)

[7.1.13 Class Hierarchies 104](#_Toc450571475)

[7.1.14 Properties 105](#_Toc450571476)

[7.1.15 Associations 105](#_Toc450571477)

[7.1.16 Property and association end hierarchies 106](#_Toc450571478)

[7.1.17 Association Classes 106](#_Toc450571479)

[7.1.2 Specific kinds of classes 107](#_Toc450571480)

[7.1.21 Context specific types and <<Classifies>> 107](#_Toc450571481)

[7.1.22 Roles 107](#_Toc450571482)

[7.1.23 Phases 108](#_Toc450571483)

[7.1.24 Quantity kinds and units 108](#_Toc450571484)

[7.1.3 Assertions about concepts 109](#_Toc450571485)

[7.1.31 Property Ownership 109](#_Toc450571486)

[7.1.32 Cardinality 109](#_Toc450571487)

[7.1.33 Global Properties 110](#_Toc450571488)

[7.1.4 Constraining properties and associations 110](#_Toc450571489)

[7.1.5 Tightening a property’s type 111](#_Toc450571490)

[7.1.6 Inferring a type from its properties 113](#_Toc450571491)

[7.2 Conceptual Modeling UML Profile 114](#_Toc450571492)

[7.2.1 UML Subset 114](#_Toc450571493)

[7.2.2 Stereotype: Classifies [Generalization, Realization] 115](#_Toc450571494)

[7.2.3 Stereotype: Conceptual Model [Package] 116](#_Toc450571495)

[7.2.4 Stereotype: Disjoint with [Dependency] 116](#_Toc450571496)

[7.2.5 Stereotype: Enumerates [Dependency] 117](#_Toc450571497)

[7.2.6 Stereotype: Equivalent with [Dependency] 117](#_Toc450571498)

[7.2.7 Stereotype: Intersection [Classifier] 118](#_Toc450571499)

[7.2.71 See also 118](#_Toc450571500)

[7.2.8 Stereotype: Is in context [Dependency] 118](#_Toc450571501)

[7.2.81 Examples 118](#_Toc450571502)

[7.2.9 Stereotype: Phase [Class] 119](#_Toc450571503)

[7.2.10 Stereotype: Quantity Kind [Classifier] 119](#_Toc450571504)

[7.2.11 Stereotype: Represents [Dependency, Generalization] 119](#_Toc450571505)

[7.2.12 Stereotype: Role [Class] 120](#_Toc450571506)

[7.2.13 Stereotype: Source Reference [Element] 121](#_Toc450571507)

[7.2.131 Tag definitions 121](#_Toc450571508)

[7.2.14 Stereotype: Sufficient [Property] 121](#_Toc450571509)

[7.2.15 Stereotype: Union [Classifier] 121](#_Toc450571510)

[7.2.16 Stereotype: Unit [Classifier] and Base Unit [Classifier] 122](#_Toc450571511)

[7.2.17 Stereotype: Value 123](#_Toc450571512)

[7.2.18 Stereotype: Has value [Dependency] 123](#_Toc450571513)

[7.3 UML Profile – Rules & Data Model Mapping Primer 125](#_Toc450571514)

[7.3.1 Representations 125](#_Toc450571515)

[7.3.2 Mapping Representation Rules 126](#_Toc450571516)

[7.3.3 Representation traversals and patterns 127](#_Toc450571517)

[7.3.4 Representation Subsets 128](#_Toc450571518)

[7.3.5 Map constraints and computations 130](#_Toc450571519)

[7.3.6 Facades and Representation Computations 130](#_Toc450571520)

[7.3.7 Property Chains 132](#_Toc450571521)

[7.4 Consistency Rules Profile 134](#_Toc450571522)

[7.4.1 Class: ExistsRule 134](#_Toc450571523)

[7.4.2 Stereotype: Facade [Classifier] 135](#_Toc450571524)

[7.4.3 Stereotype: Filter [Property, Connector, ConnectorEnd] 136](#_Toc450571525)

[7.4.4 Stereotype: Map [Connector] 137](#_Toc450571526)

[7.4.5 Stereotype: Rule Specification [Package] 137](#_Toc450571527)

[7.4.6 Stereotype: Match [Connector, Property] 138](#_Toc450571528)

[7.4.7 Class: Rule 139](#_Toc450571529)

[7.4.8 Class: Rule Computation 139](#_Toc450571530)

[7.4.9 Stereotype: Representation Rule [Structured Classifier] 139](#_Toc450571531)

[7.4.10 Stereotype: Represents [Realization, Dependency] 140](#_Toc450571532)

[7.4.11 Stereotype: Subset of [Connector End] 141](#_Toc450571533)

[8 Profile mapping to metamodel (Normative) 143](#_Toc450571534)

[9 Concept Index 145](#_Toc450571535)

Preface

**OMG**

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG’s specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG’s specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at http://www.omg.org/.

**OMG Specifications**

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG Specifications are available from the OMG website at:

*http://www.omg.org/spec*

Specifications are organized by the following categories:

Business Modeling Specifications

Middleware Specifications

1. CORBA/IIOP
2. Data Distribution Services
3. Specialized CORBA

IDL/Language Mapping Specifications

Modeling and Metadata Specifications

1. UML, MOF, CWM, XMI
2. UML Profile

Modernization Specifications

Platform Independent Model (PIM), Platform Specific Model (PSM), Interface Specifications

1. CORBAServices
2. CORBAFacilities

CORBA Embedded Intelligence Specifications

CORBA Security Specifications

**OMG Domain Specifications**

**Signal and Image Processing Specifications**

All of OMG’s formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters  
109 Highland Avenue  
Needham, MA 02494  
USA  
Tel: +1-781-444-0404  
Fax: +1-781-444-0320  
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult http://www.iso.org

**Typographical Conventions**

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

**Helvetica/Arial - 10 pt. Bold:** OMG Interface Definition Language (OMG IDL) and syntax elements.

**Courier - 10 pt. Bold:** Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

NOTE: Terms that appear in italics are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

**Issues**

The reader is encouraged to report any technical or editing issues/problems with this specification to <http://www.omg.org/report_issue.htm>.

# Scope

## Business Need

Our ability to share, manage, analyze, communicate and act upon information is at the foundation of the modern enterprise and open, collaborative government. Information sharing is essential for an integrated approach to enterprise supply chains, fighting terrorism, business and government intelligence, inter-organizational collaboration and integrating enterprise applications. Yet, this essential capability has remained difficult and expensive to achieve in information systems which are frequently isolated, stove piped, and difficult to integrate. The inability of our systems to share information hampers the ability of our organizations to collaborate and for our processes, services, and information resources to work together. Much of our information technology budgets are consumed by attempts to overcome this “semantic friction” in our systems and organizations are currently spending more on application integration than on building new applications [Gartner2011]. The overall human and financial cost to society from our failure to share and reuse information is many times the cost of the systems’ operation and maintenance.

In general, information sharing can be understood at a number of different levels.

* *Infrastructure* is the technology used to maintain data and move it from one place to another.
* *Format* is the way data are structured.
* *Semantics* deals with how data is interpreted as meaningful information. For an information system, this interpretation is reflected in how the data is processed in order to carry out the business purpose of the system.

We are effective at dealing with data infrastructure today, and we are somewhat effective at handling multiple data formats, albeit via manual and point-to-point integrations. However, we are not very good at understanding how the semantics of data in independent data sources are related. Too often, how each system interprets shared data is implicit in the specific design and operation of the systems. Differences in structure, terminology, viewpoint, and notations make system-specific data structures hard to integrate, negatively impacting the capability to federate these systems.

Full semantic integration requires information systems to all properly and consistently interpret the data exchanged among the systems. This, in turn, requires that there be an explicit understanding of what the desired semantic interpretation *is* at a business level. A semantic *model* can be used to express this understanding in a way that can be validated by the business stakeholders of the systems being integrated. And, given a formal underpinning for such a model, it can then also be used for supporting analyses and deductions necessary to carry out the necessary integration.

Unfortunately, for most existing information systems, the desired semantics have not been properly modeled. The following are some scenarios in which semantic integration is, nevertheless, critical. Diverse and disparate efforts are currently being made to address these scenarios, examples of which are included with the scenario descriptions below. But, as of today, there is no consistent way to address modeling for semantic integration in general across all these areas.

* *Data integration between business systems.* Many large businesses have a critical need to better integrate systems in support of complex products. Not only may their business area have suffered financial distress, but there may be a need for new government reporting or new analytics and integration due to acquisitions. Such organizations typically have multiple layers of existing data bases, middleware specifications and XML schemas for use in web services, event brokers, etc. Most, if not all, of the existing systems and technologies still need to be supported. There may be dozens or even hundreds of enterprise systems involved and hundreds or thousands of small applications and spreadsheets.

*Example.* A common approach chosen for integrating major business systems is to create a “canonical model” of the domain and then map data into and out of that model using data mapping tools. Unfortunately, while there are various proprietary tools to support such an effort, there is no widely available standard-based tooling for the job. For instance, while UML can be and is used for the modeling part of the job, a general modeling notation such as UML is far from ideal for the conceptual level of modeling required, and there is currently no standard profile to adapt it to the task nor for mapping data into and out of a canonical model in general. (The Model Driven Message Interoperability specification provides some support for the latter, but only limited to message format transformation for the financial services domain.)

* *Data federation across multi-disciplinary teams.* Developing complex systems often involves many parties who are widely distributed in location and time. Such development therefore requires efficient and effective information exchange during the complete development and operations lifecycle of the system. This can only be achieved by realizing semantic integration between all involved parties.

*Example.* The European Cooperation for Space Standardization (ECSS)[[1]](#footnote-1) addresses this issue by introducing the concept of *a global conceptual model.* This model is used in the implementation of “space system data repositories” as federations of physical databases. These databases are geographically dispersed and change over time but are logically integrated in an interoperable architecture, so that data can be exchanged effectively and reliably. Such data repositories need to be stable over a long period of time, so modeling must be at the semantic level *independent* of technology and tools. This modeling allows for upgrading the implementation technology without changing the model and data itself. The primary aim of this is to substantially reduce the system development and operation costs while achieving greater precision and federation.

* *Information federation across an industry.* Entire major industries, such as finance and telecommunications, need to deal with the representation of information relative to multiple contexts, taking into account different business processes, specific modeling goals and needs, visualization and implementation requirements or the existence of overlapping modeling domains. These differing contexts and conditions may require emphasizing different aspects and characteristics of essentially the same information. The representation of a concept in one view may be different from the representation of the same concept in another view as the context-specific details that are relevant differ from view to view. Information can be described using different yet compatible paradigms (e.g., domain-specific languages vs. UML and profiles) yet the meaning and semantics of the information should stay the same regardless of the format. This, again, highlights the need to focus on a common core model of shared semantic concepts.

*Examples.* Some examples of efforts to deal with industry-level information federation are the Shared Information and Data (SID) Model, developed by the TM Forum [TMForum], the Common Information Model (CIM) developed by the Distributed Management Task Force [DMTF] and the Reference Information Model (RIM)developed by Health Level Seven [HL7].

* *Information sharing and federation of threat and risk information,* Threats and risks are increasingly multi-dimensional in nature – spanning physical space and cyber space. Threat actors understand and exploit our stove piped approach to sharing and analyzing information which leads to ineffective collaboration and mediation. Only by federating information across multiple domains such as cyber, physical, critical infrastructure, criminal, intelligence and defense, irrespective of technical and political boundaries, can we effectively counter multi-dimensional intentional threats, natural events and system failures.

*Examples.* Attacks on our critical infrastructure have and will combine cyber attacks with physical attacks. This has been seen in exploits of our electric power grid where physical weaknesses are combined with Cyber to harm our physical infrastructure. By combining Cyber, criminal and terrorist information we will be better able to deal with these critical threats.

* *Data federation across government organizations.* Information sharing has been recognized by governments as a key enabler for purposes as diverse as fighting terrorism to financial transactions. There has been some progress in standardizing exchange schemas, which is a big step ahead of no standards at all, but the need exists to ensure that there is no ambiguity in the semantics of the exchanged data in order to safely enable the reuse of that data. In addition, any such standard must accept that there are and will be other such standards and that these also need to be federated.

*Example.* The U.S. Information Sharing Environment (ISE) “provides analysts, operators, and investigators with integrated and synthesized terrorism, weapons of mass destruction, and homeland security information needed to enhance national security and help keep our people safe” [ISE]. ISE depends on fixed schemas for information sharing, i.e., the National Information Exchange Model (NIEM) and the Universal Core (UCORE). These schemas provide XML Schema definitions that are claimed to be sufficiently common and universally understood by relevant stakeholders regardless of the IT systems being used within their intended domains. Even within NIEM, though, hundreds of overlapping schemas have been defined.

* *Model federation across different modeling metamodels.* The OMG itself has multiple standards related to modeling. These standards were originally created independently, resulting in difficulties when users try to use them together to share information embodied in models using the different standards. A conceptual model abstracting from the existing OMG modeling standards, would facilitate their comparison, acknowledging the commonality (or lack thereof) between the different concepts and definitions and bridging those concepts.

*Example.* OMG specifications related to just process modeling include BPMN, UML Activities, BPDM, and SPEM. A case in point in the difficulty this has caused relates to the *UML Profile for DODAF and MoDAF* (UPDM), a wide ranging profile supporting US Department of Defense (DOD) and UK Ministry of Defence (MOD) architecture frameworks. The UPDM community wishes, for example, to be able to use BPMN process models in the context of their UML Profile. A stopgap tactic has been to define an additional *UML Profile for BPMN*, which allows BPMN-looking diagrams to be drawn in UML, but it is clear this is not a strategic approach. A better approach would be to create a “process modeling” conceptual domain model that would then permit model bridging relations between BPMN, UML, BPDM and SPEM models, allowing sharing across users’ process models

* *Schema Evolution*. As information systems evolve to support changing enterprise needs, the datasets they use need to evolve as well. While some changes are additive and readily accommodated, others involve factoring and evolving concepts. At their core, such changes require the evolution of the dataset schema underlying the system and the migration of the data from the old to new schemas. Such changes also impact the logic that interacts with the dataset and every external interface and related data structure. While there is some tooling available for schema migration, there is little available to aid in the evolution of the logic and external interfaces. The absence of semantic understanding of the relationship between the schema and external interface data structures makes tooling to aid in the evolution problematic.

*Example.* It is common for an enterprise to represent the concept of *customer* as a composite of information about the person and the role that person plays with respect to the enterprise. Evolving needs, including regulatory requirements, require many enterprises to now factor this concept so that they can represent that the same person may play other roles as well, such as employee. Such semantic understanding is required to enforce constraints such as a prohibition against the same individual playing both the customer and employee role in a transaction. The absence of semantics-based tooling makes such changes labor intensive and error prone.

Current standards for information and data modeling may be effective at defining a particular data model for a particular application using a particular technology to solve a particular problem. But, as highlighted by the above examples, the methodology for using these standards at a higher level of abstraction – namely for cross-domain and cross-organizational semantic modeling – is not as well or as widely understood. As a consequence, the models available within a given organizational context are often not well suited to application across multiple dimensions or technologies, and so poorly support the needs for sharing and federation.

## Approach

As described in Section 6.2.1, the scope of SIMF encompasses conceptual domain modeling, logical information modeling and the modeling of bridging relations between models at all levels. Figure 1 summarizes the general organization of information models into conceptual, logical and physical layers and indicates the scope of SIMF within it. The following more precisely defines the terminology used to describe this scope.

* *Conceptual Domain Model (CDM).* A CDM is not a traditional data model, as such, but, rather, a model of the terms and concepts of an area of concern or *domain*[[2]](#footnote-2)*,* which may be a broad industry area such as telecommunications, finance or even metamodeling or may be more focused on a specific application area. From a formal point of view a CDM is a model of a real or possible world. It primarily addresses the semantics, concepts and terminology of a domain, capturing the meaning that usually is not available in a data model, while abstracting out data representation, viewpoint and application specific considerations. The objective of a CDM is to capture the semantics of one or more domains as a well-defined set of (potentially federated) concepts, predicates (to express properties about the concepts and to relate them) and integrity rules (constraining instances). For a given domain, many CDMs may co-exist, e.g. CDMs that have been developed by different entities and express differing conceptual frameworks.

For the purpose of this RFP, conceptual domain modeling is limited to modeling the information concerns of a domain as a conceptual domain model. Modeling processes and services is considered out of scope. However, this does not imply that all modeling of the dynamics of concepts is necessarily out of scope. (Future RFPs may address further process-oriented conceptual modeling requirements.)

* *Logical Information Model (LIM).* A LIM acts as an intermediary between CDMs and physical data schema (see below). The objective of a LIM is to provide a purpose-specific but implementation technology-independent view of information in terms of logical data structures. There can be multiple different ways to represent the same information from different viewpoints and for specific purposes. Each viewpoint may have its own structure, local vocabulary and subset of all possible information in a domain. These purpose specific commitments are made in the LIM. Elements of a LIM are related to the CDM concepts, predicates and integrity rules they represent (using Model Bridging Relations, see below) and may extend or embed other logical elements. A LIM model addresses a specific viewpoint and purpose and, as such, selects those types, properties and relations of interest and structures them for that purpose.
* *Physical Data Schema (PDS).* A PDS describes how to implement a LIM in a database or exchange format of choice. That is, it defines the application- and technology-specific representations of data. There can be many PDS representations of the same LIM. PDSs grounded in LIMs (using Model Bridging Relations, see below) provide the basis for federation of data defined in those schemas. Such fixed schemas become a particular *projection* of information for a particular purpose, but not the only way to access the same information.
* *Model Bridging Relation (MBR).* An MBR defines a connection between different sets of elements in the same or different models. This connection may be between models across the conceptual, logical and physical modeling layers, or within models of a given layer. MBRs may bridge different models created as part of a single wider effort or they may address connections between independently conceived models. Linking the semantics of information in its different conceptual, logical and physical representations, MBRs are the foundation of federation.



Figure 1 Information Modeling Layers and SIMF Modeling Scope

As indicated in Figure 1, the scope of SIMF includes CDMs, LIMs and MBRs (where the lines in the diagram represent MBRs). While PDSs are out of scope for SIMF, model bridging relations to PDSs are part of SIMF.

## Unified Meta Model & Notation

While SIMF supports modeling at multiple levels, there is a single meta model for SIMF. A model defines the abstraction layer it is intended to model. Likewise, the same syntax is used at all levels. While there is a unified meta model, some model elements are more appropriate for one layer or another – which is defined as an appendix.

# Conformance

The Conformance clause identifies which clauses of the specification are mandatory (or conditionally mandatory) and which are optional in order for an implementation to claim conformance to the specification.

There are five distinct types of conformance. These are listed below. Unless otherwise stated these types of conformance are independent.

1. Abstract syntax conformance. A tool demonstrating abstract syntax conformance provides a user interface and/or API that enables instances of concrete SIMF metaclasses to be created, read, updated, and deleted. The tool must also provide a way to validate the well-formedness of models that corresponds to the constraints defined in the SIMF metamodel.
2. UML Profile conformance. A tool demonstrating UML Profile conformance provides a user interface and/or API that enables instances of SIMF UML notation to be created, read, updated, and deleted. Note that a conforming tool may provide the ability to create, read, update and delete additional diagrams and notational elements that are not defined in SIMF.
3. Model interchange conformance. A tool demonstrating model interchange conformance can import and export conformant XMI for all valid SIMF models, including models with profiles defined and/or applied. Model interchange conformance implies abstract syntax conformance. A conforming SIMF tool shall be able to load and save XMI in SIMF format.
4. Semantic conformance. A tool demonstrating semantic conformance provides a demonstrable way to interpret SIMF semantics, e.g., data transformers, code generation, model execution, or semantic model analysis.

Where the SIMF specification provides options for a conforming tool, these are explicitly stated in the specification. In a number of other cases, certain aspects of the semantics are listed as "undefined" or “intentionally not specified” or “not specified”, allowing for domain- or application-specific customizations. Only customizations that do not contradict the provisions of this specification will be deemed to conform to it. However, models whose meaning is based on such customizations can only be interchanged without loss with tools that support the same or compatible customizations.

This specification comprises this document together with XMI serialization contained in machine-consumable files as listed on the cover page. If there are any conflicts between this document and the machine-consumable files, the machine-consumable files take precedence.

# References

**3.1 Normative References**

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

List of normative references.

* UML
* SMOF
* RDF
* OWL
* ODM
* PRR
* BFO
* JCGM 200:2008

**3.2 Non-normative References**

List of non-normative references.

# Terms and Definitions

For the purposes of this specification, the following terms and definitions apply.

See section 1.2

# Symbols

List of symbols/abbreviations.

# Additional Information

## Changes to Adopted OMG Specifications [optional]

None

## 

## 6.2 Acknowledgments

The following companies submitted this specification:

* Data Access Technologies, Inc. (Model Driven Solutions Division)
  + Cory Casanave
* Nomagic, Inc.
  + Jim Logan
* PNA-Group, Ltd.
  + Sjir Nijssen
* 88 Solutions
  + Manfred Koethe
* Thematix Partners LLC
  + Elisa Kendall

The following companies supported this specification:

1. Tibco Software Inc.
2. Paul Brown

# Introduction to SIMF Concepts

The following is a high-level, non-normative, description of some of the fundamental SIMF concepts.

The fundamental concepts will be described in a way that most practitioners can easily relate it to their familiar experiences. In this chapter we will gradually build a semantic-conceptual architecture (an architecture that is completely independent of any particular technology and in which there is a clear distinction between the world of the things and the world of the names (representations)).

The prime aim of this chapter is to demonstrate the value that a SIMF semantic-conceptual model and transformations can offer to the ever increasing need of federation of (information) systems in business and government practice. The parentheses are placed intentionally such that SIMF can also be of value to the emerging MBSE (Model Based System Engineering) discipline, federation of risk and threat information as well as the emerging discipline of legal engineering as an aid to produce person based services.

## *Pragmatic world view*

Any conceptual system has strengths and weaknesses and is colored by the influences and domains which influenced the conceptualizations. From a pragmatic point of view there is a good understanding of what a “Tree” is (the word sense of tree that refers to a plant most babies could identify). However, when you look very close and at the “edges” such categorizations can be fuzzy. On a similar note when you get to the edges of quantum mechanics and near-light speed travel, our common views of the world can break down, or become unknown or controversial. Those who study language carefully may say that every meaning of every word is contextual and that dictionary and thesaurus don’t completely work.

While all of the above may be true, taken to the extreme it would make communication impossible and synthesizing or transforming information impractical. On the other hand, if you take a pragmatic world view there is a lot that can be done to communicate, even across those with different backgrounds, and most of us just don’t deal with quantum mechanics and “warp speed”. SIMF and the conceptual models built on it use a pragmatic world view and are designed to deal with structured (or at least semi-structured) data. However, such pragmatics are modularized into component models such that those who need to deal with extreme detail in some area may do so without breaking the entire system.

If the “precision dial” were turned all the way up to account for all these edge cases the model would get complex and unusable.

The other extreme is to make definitions and their representation very narrow and specific, perhaps to a system or particular discipline. The other extreme may also have such “fuzzy” definitions that they are no more than hints. This extreme also has its problems and would make wide-scale communications and synthesis impossible. In fact, such overly specific data representations and fuzzy schema are essentially what we are federating with SIMF – the design of specific systems.

So between overly complex and overly specific or fuzzy is the useful middle ground we seek. That middle ground is to be precise and true within this pragmatic world view.

SIMF does not restrict a model to being very loose or very precise, but in general we expect:

* Terms mean what they do in a dictionary definition, refined to a word sense
* Where a concept is more specific it will be scoped by appropriate context (e.g. an expert’s quantification [Keslick, John A. (2004). ["Tree Biology Dictionary"](http://www.treedictionary.com/). Retrieved 2012-07-30.] of what defines a tree may be more specific than common concepts)
* Concepts will have the most general definition and relationships reasonable within the context in which they are defined (note that this sometimes introduces general abstract concepts). Note that this is different from common practices in defining schema where the same essential meaning is repeated everywhere it is used.
* We are treating the world in terms of Newtonian mechanics (e.g. space/time is not relative)
* Where applicable, we are concerned with the surface of the earth and the near vicinity (e.g. in understanding location and gravity). However, such assumptions should reference an earth context and off-earth models are possible.
* Generic concepts are better under-defined (having fewer axioms than may be possible) than over-specified. This allows more specific concepts or “theories of the world” to specialize and reuse the generic ones.
* The concepts defined and mapped are being used in some structured way, not unstructured natural language (this does not preclude the structuring of natural language by automated or “human in the loop” means, but this is out of our scope). We do expect that natural language processes may interpret text to define SIMF axioms.

SIMF does provide mechanisms to support the evaluation of context, scope and trust so domains “on the edges” may still use SIMF, but should be careful about what SIMF features or SIMF models may say about common concepts.

In terms of the spectrum of formal logics and formal languages, SIMF is in the middle. It is specifically designed and intended for the role of semantic mediation or real (and probably messy) data. This is different from most of the “ontology languages” that are designed for general inference and “decidability” of 100% valid and consistent formal models. The priority for SIMF conceptual models is that they capture the real domain semantics, even if those semantics may be inconvenient for some inference algorithms. As such, the SIMF logic does not fit into some of the common categories; it is not a “description logic”, nor is it a “frame logic” or “first order”. In particular, SIMF relationships are “first class”, that is statements can say things about other statements like “Jim doesn’t believe joe likes sue”. Each of these relationships is its own thing, not part of any “frame” or “object”. SIMF includes a specific concept of context to scope statements. Also, you can also create new kinds of kinds – “metatypes” in modeling land. These capabilities are not arbitrary – they represent real concepts people need to express about the real world. By expressing those concepts directly we have a valid “pivot point” for federating data. It should be noted that the kernel of the SIMF language is formalized -described in formal logic, but SIMF does not require such formality in conceptual or information models.

## *Models*

SIMF models describe some subject area or domain for the purpose of defining and sharing *information* within and about that domain and with other related domains. SIMF defines a modeling language for this purpose; users of SIMF use that modeling language to describe applicable concepts; in some cases, a model captures generic, cross domain concepts while in the remaining cases a model may be more specific, that is specific to a discipline, domain, organization or system (we will consider all of these a “domain”). Ultimately a model focused on a domain, any domain, shares many concepts with other domains – the sharing of such concepts is the basis for communications among people, organizations and systems with different backgrounds and purposes. For this reason, domain specific models typically use or specialize more generic cross-domain concepts. To support this requirement SIMF also provides for models that describe general concepts and then provides capabilities to map those general concepts to more specific domain concepts.

**Information**

Models are themselves *information* **about something**. A concept defined in a model is not “the thing”, but a **description of that thing from some perspective**. For this reason the subject of the model and the perspective are important for understanding the content of the model and how it may relate to other models. A model is fundamentally *information*, that information may be represented in different ways: textually, as tables, graphically, etc. SIMF defines some ways to represent information but other representations may be used as is appropriate for a purpose. The same information is typically rendered in multiple different formats for different purposes and stakeholders.

**Bridging technologies**

A model needs some way to store information. SIMF defines an abstract syntax (nevertheless illustrated with sufficient concrete examples) for storing SIMF information and multiple physical representations, in keeping with the SIMF philosophy of *bridging* technologies. Regardless of the technology, a SIMF model can be thought of as a repository of information. Since SIMF is, its self, information about information it is also called a “meta model”.

## *Concepts*

Everything we describe in a SIMF model is considered a *concept*. The term concept is used because a model describes how we conceive or understand something from a perspective. For example, we may all understand there is something we call the “Eiffel tower”. In the figure below this information is represented in a very familiar format, an extended Venn diagram. We see a clear distinction as said above between the world of the things and the world of names and information about the things, an essential distinction in SIMF.

A person preferring a textual representation could use the following formal representation:

(1a) “There is something in the world that we are thinking of when the term “Eiffel tower” is used”.

Or in short but sufficient format:

(1b) There is something in the world that is referred to as Eiffel tower.



Figure 2

But how we think about it (the thing designated by the term “Eiffel tower”) or describe it may be different. Differences aside, there is something in the world that we are thinking of when the term “Eiffel tower” is used so we want to connect these different descriptions of the same thing – it is the “real thing” that connects us even though there is no way to have that real thing in our minds or models. We connect the dots through our concepts grounded in our mutual understanding of the world.

Concepts in SIMF may be about anything: “Real things” like the Eiffel Tower, Imaginary things like Unicorns, types of things, characteristics of things, relationships between things, rules, even names. In SIMF “Concept” is the root of everything else.

Once you define a concept in a model you can start adding to it, saying more about it – of course each thing you add is a new concept. A SIMF model becomes a connected graph of these concepts and concepts about concepts. Some of those concepts will also come from other models.

In Figure 3 quite a few additional pieces of information are added to say more about the Eiffel tower. We also bring in a second example, in this case the well-known Colosseum. Please note that the English term Colosseum identifies the same thing as the French term Colisée. Figure 3 can be used to illustrate the essential concepts of context and perspective in a way that practitioners easily understand.



Figure 3

Additional detail on terms and concepts can be found here: <https://github.com/ModelDriven/SIMF/blob/master/SupportingDocuments/PNA-Group/1981-10%20An%20architecture%20for%20knowledge%20base%20software-96-pages.pdf>

### Dictionary Concepts

In many modeling languages you need to know a lot about something to define it in a model. In SIMF, you can invent new concepts any time you like as a dictionary concept. The precise description of this far reaching aspect will be discussed in a separate section. In defining such a new concept you may give it a name and it is best to give it a description. Other concepts are defined more contextually and may not require a name. For example, the concept that the Eiffel Tower was erected in 1889AD or that it is 324 meters tall (including the TV antenna) do not need extra names; they are just facts about the Eiffel tower. Facts are concepts describing one or more other concepts. In the figures above, each line is a fact. However, facts do have “identity” so their provenance can be traced, crucial in trusting any “fact”.

Dictionary concepts may also have qualifiers that define the plurality (e.g. singular or plural), timeframe (past, present or future) and kind of concept (noun, verb, etc.). Dictionary concepts can also be related to one-another in several ways and provide the basis for defining information elements.

## *Information, Values ands Anything*

Values, such as the number 5 or the text “Frog” are not considered concepts, they are information we use to describe concepts. Information and values forms a tree that is distinct with “concepts”. Above concepts and information is the superclass “Anything”. All elements in SIMF derive from “Anything”

## *Context*

In this section we will use Figure 4 as the diagrammatic representation to illustrate contexts and perspectives.



Figure 4

As descriptions of something from a certain perspective, not all concepts are the same. You may trust some and not others, some may be true of a certain timeframe, and some may only be valid in specific conditions. A central problem in information federation is that the context of information from various information repositories is different. Different authorities may or may not agree on facts, they may or may not use the same or conflicting terms, they may describe different aspects of the same thing. Each description, each concept, needs to be understood within the context in which it is defined and used. SIMF defines “context” as a “first class” concept – that means that dealing with context is part of using SIMF.

In SIMF, a context connects a set of facts with a set of things those facts applies to. The context *contextualizes* a set of concepts that it applies to. Facts and rules are defined in one or more context and apply to everything those contexts contextualize.

There is a fact that the Eiffel Tower is the tallest building in Paris. Who said that? Can you believe it? If I got that fact from Wikipedia, does that make it believable? The source of information is one of the contextual dimensions. What if a taller building were built in 2020? The fact that “the Eiffel Tower is the tallest building in Paris” would be true in the timeframe from 1889-2020. Timeframe can also contextualize a fact. From this example we can see that the applicability and interpretation of a fact is contextual and that multiple contexts may be applicable to any one fact. Your perspective defines what contexts are meaningful and trusted in a situation. SIMF defines context and provides for the provenance of information, the interpretation of that information to form “trust” and “belief” is outside of this specification. Tools that build on SIMF may provide was to evaluate or quantify trust and belief based on the information SIMF provides.

## *Entities*

Individual things that exist in our world, or some conceived world, are entities. Examples of entities include Barack Obama, The Eiffel Tower, Hurricane Andrew, an agreement between Barack Obama and the country France, the mortgage for my house, etc. Entities include physical things (with a “special/temporal extent) that do exist, may exist in the future or have existed in the past. Entities also include specific things we conceive, such as agreements between specific individuals. Entities have a “lifetime”, and they may change over that lifetime however that lifetime may or may not be known and we may have facts about an entity before, during and after its lifetime

## *Situations*

Various configurations and properties of things exist over some timeframe, as understood from some perspective. Such configurations of things are *situations*. A situation is a description of any configuration of things, including the logical, temporal and physical relationships between them. Situations can be large and comprehensive – like our solar system or as granular as your weight at a particular time. Facts about anything can be contextualized by a situation and that situation may, in turn, be contextualized – perhaps by a timeframe. Situations may be real or imagined, in the past, present or future.

Situations form the basis for relationships and facts in SIMF as any fact or relationship is bound by a time dimension when it was true. As all situations are bound by some timeframe where they are true (regardless of whather that timeframe is known or not), situations are the link between “things” and time.

## Relations

Relations (a kind of situation) are concepts that logically connect other concepts and represent the primary semantic elements of SIMF. For example, the height of the Eiffel Tower is a “Height” relation between the Eiffel Tower and the quantity “324 meters”. That my computer is on my desk may be a “supports” relation between my desk (as the support) and my computer (as the thing supported). While the set of relations describing something may change over time, each relation is considered constant and atomic– the “parts” bound to the relation (the things they relate) don’t change, any change would be a new relation.

Relations can involve any number of “roles” that define how other concepts are connected to the relation, however there are special considerations given for the common case of a relation involving exactly 2 primary entities – binary relations.

It should also be noted that in SIMF relations are “first class”, that is relations may have relations and properties. For example, the timeframe a relation is true is a property of that relation.

## *Types*

Types categorize and describe classes of things. Everything in SIMF has at least one type and may have any number of other types as well. The set of types that categorize a concept may change over time and be specific to a context. Facts describe and may constrain the extent of any type.

### Individual Types

Most common things defined in information models are entity types – kinds of physical or conceptual individuals that exist over a specific timeframe.

### Value types

Most concepts represent things that can change over time – like the Eiffel Tower or the United States. There is another class that represent pure values that do not change and do not have a lifetime. Numbers are the most common kind of such values – you can’t “create” the concept of “5”, it “just is” based on our accepted theory of numbers. Values are typically used as the object of properties (also called attributes) such as something’s size, mass or some other quantity. Numbers, strings and certain fixed enumerated values are considered value types.

### Situation Types & Situational Roles

Situations have their own specialization of the type concept, one that defines additional types –*Roles*. A role represents the state or behavior of something involved in a particular kind of situation. For example, a person may play an “employee” role with respect to a company playing an “employer” role. The concept of the employee and employer is, in this case, specific to a particular employment relationship (a relationship is a kind of situation). Each situation type defines the roles that can be involved with that kind of situation.

Each role also defines the type of entities that can play the role. For example, the employee role can only be played by a person. Roles are bound to individuals in a situation. Roles also serve the purpose of “variables” and “arguments” in programming languages.

Since roles are types, all type relations apply to roles.

Examples:

* Barack Obama is bound to the “President” role in the Organization “United States of America” during the period January 20th, 2009 (Noon EST) to January 20th, 2016 (Noon EST).
* The “Eiffel Tower” is bound to the “subject” role and “324 meters” is bound to the “object” role in the directed relation “Height”.

**Note on naming roles**: In UML it is best practice to name association ends (roles). In SIMF such names are optional. If the role adds no new semantics (other than the involvement with the situation or relation) the name can be inferred from the name of the type that plays the role – it is <type> in the context of the situation. Forcing users to make up more names than is necessary is confusing and increases the terms that have to be understood, thus we only want to name unique concepts.

### Relation Types

Relation types are a specific kind of situation type that represents a semantic connection between the involved roles. Relations are atomic semantic units that cannot be further decomposed or changed. A change in the semantic relationship between concepts requires a new relationship, perhaps of the same type.

## Rules

Rules define something that must be considered true within a context (including situations, types and relations). There are various kinds of rules, some of which are defined in SIMF and some which are defined in SIMF models. Rules may have an <initial situation> which is a template to set up any preconditions for the rule and <apply to> a set of context.

### Production Rules

A production rule computes a set of inferred relations based on a template when the rules <condition> is TRUE. If <implicit> is true the rule is unconditional and will “fire” whenever the condition is TRUE, otherwise the rule must be explicitly invoked.

### As-A Rule

An As-A rule provides for an instance of a type to be viewed as (or coerced to) another type when an instance of the <source type> is bound to a role requiring the <destination type>. As-a rules provides for a temporary façade over the instance, which may be useful as a bridging relation. The As-A rule implies the <destination type> and the <implies> template.

### Logical Grounding Rules

A logical grounding defines one context<applies to> (which is frequently a relation type) as defined by a set of more basic relations <implies> where <condition> is TRUE for <applies to>.

### Constraints

Constraints define some condition that must be TRUE, else the model is not valid. A constraint may be enforced for all time (at once = false) or at a particular instant in time elements (at once = true). SIMF does not infer any elements to attempt to satisfy a constraint, such inferences are done with production rules or by external systems.

Constraints are evaluated at the end of a transaction.

### General Constraints

A general constraint has a constraint expression (condition) that must evaluate to TRUE for the constraint to be satisfied.

### Multiplicity Constraints

A multiplicity defines the maximum or minimum number of relations between two types (normally roles) within the context (normally a relation) the rule applies to.

### Uniqueness Constraints

A uniqueness constraint identifies a set of roles. The values of these rules taken as a set must be unique within the context the rule applies to.

## Quantifiers

When expressing rules it is frequently required to express relations ranging across all or some instances of a type. Specific roles may be defined that range over (quantify) a type. The specific kinds of quantifiers are:

* All – all instances of the type.
* There exists – at least one instance of the type.
* One of – exactly one instance of the type.
* Most – An extension of “there exists” which applies in most situations, but not all. This can be used for defaults.

Quantifiers may be used to make statements about all elements of a type. For example: most teenage boys like supermodels.

## Expressions

SIMF provides for general expression which are then used in other parts of SIMF, for example in rules. The SIMF expression meta model is minimal, the bulk of the functionality to be provided by function libraries. The built-in capabilities are:

* Constant reference
* Role value
* Traversal of a relation
* Function Call

The philosophy of SIMF expressions is to put almost all expression semantics into functions. A function library is anticipated but not yet defined.

## Properties & Units

Properties are values that describe a concept as a quantity or other value. Other terms for Properties are “Characteristics” or “Attributes”. To provide for a semantic definition, properties always have a domain-relevant type. In the case of quantities (a property that can be identified with a number and a Unit), there is always a unit corresponding to some “quantity kind” such as temperature or mass. Units are, of course, critical in understanding what a property really means. Properties are defined with “Property types” that are a kind of directed relation. The “object” of the directed relation for a characteristic is always a value type.

Based on the definitions in JCGM 200:2008 : A “Quantity Property Type” (e.g. Height) has a range of a “Quantity Kind” (e.g. Length) which is represented by a “Measurement Unit” (e.g. Meters) . An instance of a Measurement Unit is a “Quantity Value” (e.g. 324 Meters).

Property values that are not quantities (having a unit) may have any value type as their object type.

A property an unit library is currently defined in the threat and risk submission and will be moved to SIMF.

## Naming

A SIMF concept can have any number of names. A “Term” binds a string label to a concept in some context. A term has a “Priority” to define how preferred that name is in some context. Names may or may not be unique.

## Representations

When bridging from a conceptual model to a logical or physical model there must be ways to represent concepts. Representations are also concepts and a Representation Type is a specialization of “Type”. A representation “Represents” a concept and a Representation Type represents some other type.

While this forms the basic meta-model for representation, rules are required to specify the details of representations.

## Important Relations

### Contextualization

The “Extent” relation exists between a context and the concepts it <contextualizes>. The inverse is <in context of>, Contextualization is the basis for many other relations in SIMF, including the relation between a type and the instances it <categorizes>.

### Equivalence

As has been explained previously, a concept describes something or some set of things. There can be many such descriptions in different context. An *equivalence* relation states that a set of concepts describe exactly the same thing or set of things – they are different concepts of or assertions about the same thing “in the world”.

**Equivalence and open world:** Equivalence is one mechanism that SIMF uses to support an “open world”, that is that any model can assert anything about anything anywhere. This is accomplished by defining concepts of the same thing in different context and stating that they are equivalent.

**Equivalence and consistency**: Equivalence in SIMF does not assert consistency as do some logical languages; different descriptions of the same thing may or may not be consistent. A SIMF tool may be able to determine if two contexts are consistent.

### Generalization

Generalization defines one context (including situations and types) as specializing another and, conversely, that one context generalizes the other. The set of concepts contextualized by the specialized context must be the same as or a subset of the concepts contextualized by the general concept. Further, any rules defined for the generalized concept must apply to the more specialized concept.

Generalization is typically applied between types as subtypes, supertypes. However any context can participate in a generalization.

## SIMF Lexical Scope & Physical Representations

A SIMF model is an information resource and as such has structure and semantics as defined in the SIMF abstract syntax. Only a small subset of that abstract syntax is relevant to the structure of the information, this structure is defined in the “Lexical Scope” portion of the model. Only elements stereotyped as “Structural” are part of the logical information structure, all other elements are represented using that basic foundation. The lexical scope structure is than mapped to various technology formats such as XMI and RDF.

Like most modeling languages, a SIMF model is “block structured”, that is certain elements contain other elements. The model structure is one form of context that may or may not be relevant to its interpretation, SIMF attaches no domain semantics to model structure. Most users will not be concerned with the details of the lexical structure, but an understanding of it is important for implementers, as follows…

A key element is the abstract “*Lexical Scope*” which can reference and/or import any context in any model. Referencing a context makes it visible within the lexical scope, importing a context asserts that external scope within the lexical scope. A lexical scope may also define Terms, which provide labels for concepts.

A *Model* is a lexical scope with no parent, it is an independent data resource.. A model is identified as modeling a particular level of abstraction (Conceptual, Logical, Physical or Binding) and subject area.

A Block (including its subtypes “Type” and “Situation”) are substypes of Lexical Scope and owned by a lexical scope. A block owned by a lexical scope is asserted by that scope. Blocks are similar to UML packages or statements within braces in Java.

Bindings bind an individual to a type (or role) in a situation. Each “end” of an instance of a relation is a binding.

Bindings are possible due to SIMF concept identity. Every concept in SIMF has identity: Every individual, relation, binding and type. For this reason SIMF relations may be made between anything. It may be noted that this is not “first order”, as in “first order logic” (FOL). Supporting a rich conceptual model and mappings requires concepts that are not first order. Portions of SIMF models may be mapped to first order languages to take advantage of their inference capabilities.

**Type coercion**: The referenced or local bound concept may or may not have a type compatible with the role it is playing. If it doesn’t and there is an As-A rule for coercing a type of the referenced concept to the requirements of the role, the As-A rule will fire to create a local “façade” for the concept of the appropriate type. This type coercion is one of SIMF’s model bridging capabilities.

### Transactions

SIMF is intended to support information in multiple repositories using diverse schema being federated. As part of that federation the concept of a Transaction is defined. Any change in a SIMF model visible to a SIMF implementation will generate a transaction. Each transaction will reference the elements created or invalidated by that transaction. A transaction may also spawn sub-transactions, some of which may be the result of rules that cause other changes. Every element of information visible to SIMF implementations can then be tracked to its source.

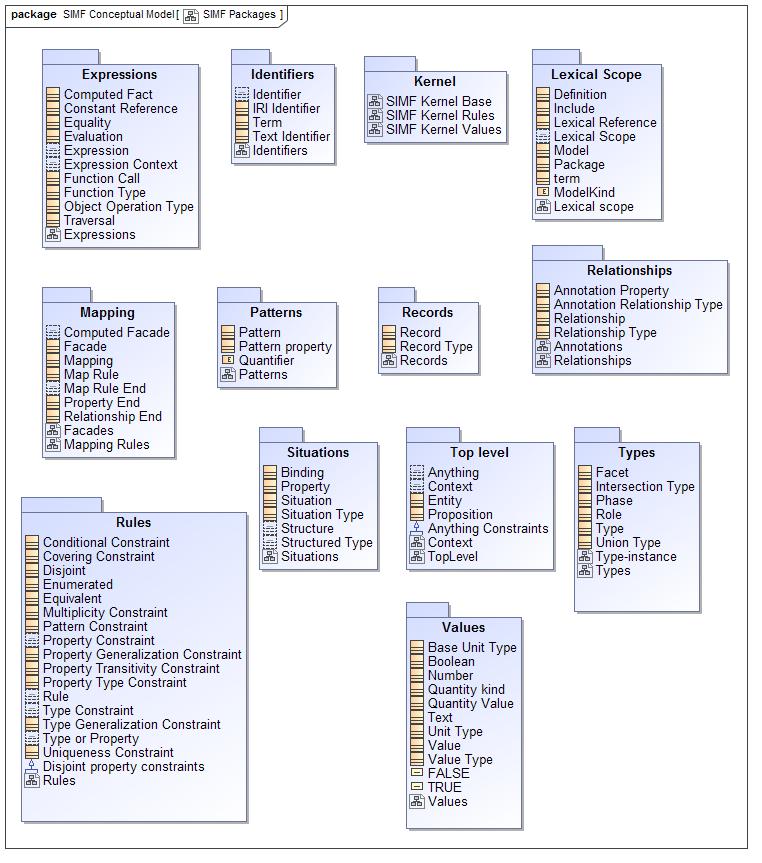
Transactions also assist in integration. It is not the intent that SIMF replicate every semantic of every data technology, that would be impractical. Where changes occur that would impact a data resource the changes (represented by transactions) of that impact are provided to technology adapters such that the change can be made – thus it is not expected or required that data resources would be wholesale converted to one format, changed and converted back – only changes need be propagated. However, transactions are a capability offered – it is up to implementations to define exactly how change is propagated.

# Abstract Syntax [Normative]

This section presents the normative specification for the SIMF metamodel. It begins with an overview of the metamodel structure followed by a description of each sub-package.

## SIMF Conceptual Model

### Diagram: SIMF Packages

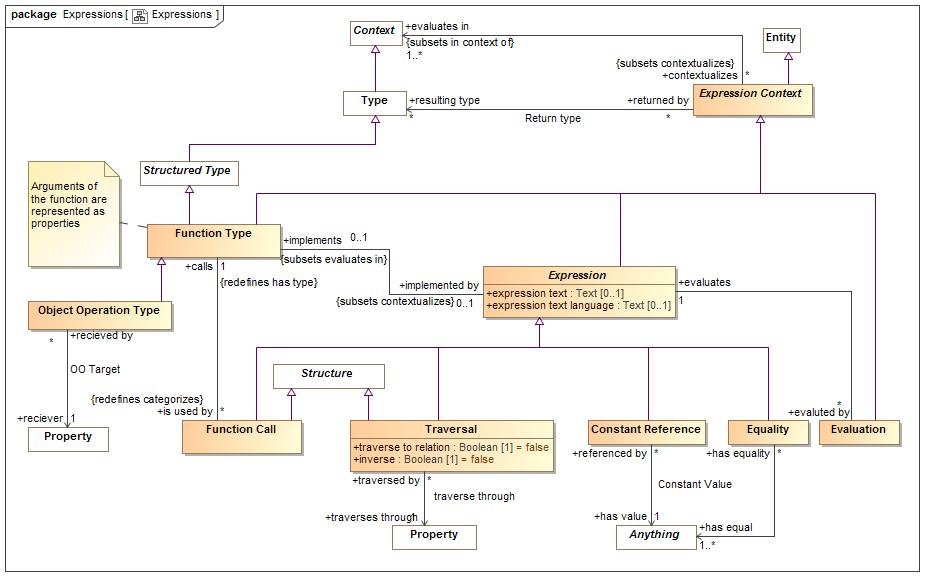


1. SIMF Packages

## SIMF Conceptual Model::Expressions

Expressions define computations across SIMF models.

### Diagram: Expressions



1. Expressions

Expressions define computations

### Class Computed Fact

The actual evaluation of an expression which may cause the creation of more assertions (i.e. representing the result of a calculation).

**package** SIMF Conceptual Model::Expressions

### Class Constant Reference

A calculation that returns a concept instance. Is of type of "has value"

#### Direct Supertypes

[Expression](#_f9bba899ada544a47c36bb071e9024f5)

**package** SIMF Conceptual Model::Expressions

#### Associations

has value : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [1]



A constant value referenced in an expression.

### Association Constant Value

**package** SIMF Conceptual Model::Expressions

#### Association Ends

has value : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [1]



A constant value referenced in an expression.

referenced by : [Constant Reference](#_f3f61859903284f1b00fc6feee0b33f8) [\*]



### Class Equality

Returns TRUE if all <has equal> have the same value or represent the same thing or set of things regardless of how they are represented.  
Equality will return TRUE or FALSE.

#### Direct Supertypes

[Expression](#_f9bba899ada544a47c36bb071e9024f5)

**package** SIMF Conceptual Model::Expressions

#### Associations

has equal : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [1..\*]



Set of things that must have the same value or represent the same thing or set of things.

### Class Evaluation

The evaluation of an expression. All references to an evaluation shall return the result of evaluating the <evaluates> expression. All expressions referenced within an evaluation shall return the result of evaluating that expression.

#### Direct Supertypes

[Expression Context](#_d847ee03faa23264a18dd452d21972fc)

**package** SIMF Conceptual Model::Expressions

#### Associations

evaluates : [Expression](#_f9bba899ada544a47c36bb071e9024f5) [1]



### Class Expression

The computation of a value which is then bound to the context from which it is called. The expression has a type of the most general type it can return.  
  
An expression may reference other elements. Where the other elements are also expressions they will be considered part of the referencing expression and evaluated in the context of that expression.

#### Direct Supertypes

[Expression Context](#_d847ee03faa23264a18dd452d21972fc)

**package** SIMF Conceptual Model::Expressions

#### Attributes

expression text : [Text](#_0e6e6fe0a29fb43221940aa4118b04a2) [0..1]



Textual expression of the expression which is further refined by subtypes of expression.

expression text language : [Text](#_0e6e6fe0a29fb43221940aa4118b04a2) [0..1]



expression language used for the expression text

#### Associations

implements : [Function Type](#_cff99d2f22ee84a9e95ea582786a897b) [0..1] *Subsets*: evaluates in:[Context](#_66d62b068053cee3464e1e03e6035eed)



Function implemented by an expression

evaluted by : [Evaluation](#_764178c56513beb91e5b5964ec31da8e) [\*]



### Association Expression Context

**package** SIMF Conceptual Model::Expressions

#### Association Ends

evaluates in : [Context](#_66d62b068053cee3464e1e03e6035eed) [1..\*]



Context of evaluation and namespace resolution for an expression.

contextualizes : [Expression Context](#_d847ee03faa23264a18dd452d21972fc) [\*]



### Class Expression Context

A static or dynamic context in which an expression is evaluated.

#### Direct Supertypes

[Entity](#_eb8398b5a178c638b98597120ec51c4d)

**package** SIMF Conceptual Model::Expressions

#### Associations

evaluates in : [Context](#_66d62b068053cee3464e1e03e6035eed) [1..\*] *Subsets*: in context of:[Context](#_66d62b068053cee3464e1e03e6035eed)



Context of evaluation and namespace resolution for an expression.

resulting type : [Type](#_dfe1514224ca21cedba7b2b29802db50) [\*]



Type of the result of a function

### Class Function Call

An element of an expression that performs some operation based on a function type and produces a result. I.e. plus(a,1).  
Arguments are bound to the function call via bindings.

#### Direct Supertypes

[Expression](#_f9bba899ada544a47c36bb071e9024f5), [Structure](#_8c517cf1950741c0f89edebf828214cc)

**package** SIMF Conceptual Model::Expressions

#### Associations

calls : [Function Type](#_cff99d2f22ee84a9e95ea582786a897b) [1] *Redefines*: has type:[Type](#_dfe1514224ca21cedba7b2b29802db50)



Function called

### Association Function Implementation

**package** SIMF Conceptual Model::Expressions

#### Association Ends

implemented by : [Expression](#_f9bba899ada544a47c36bb071e9024f5) [0..1] *Redefines*: has type: [Type](#_dfe1514224ca21cedba7b2b29802db50)



Expression which defined the implementation of a function.

implements : [Function Type](#_cff99d2f22ee84a9e95ea582786a897b) [0..1] *Redefines*: has type: [Type](#_dfe1514224ca21cedba7b2b29802db50)



Function implemented by an expression

### Class Function Type

A kind of function which performs a calculation on arguments (properties) to produce a result (function result). I.e. the definition of plus().  
  
Functions are intended to be side-effect free and context free (they only depend on their arguments and don't change anything) but assertions to specify that certain functions are pure may be required,  
FUNCTION ARGUMENTS ARE PROPERTIES.

#### Direct Supertypes

[Expression Context](#_d847ee03faa23264a18dd452d21972fc), [Structured Type](#_50241f5936e61055293ca95f860768d8)

**package** SIMF Conceptual Model::Expressions

#### Associations

is used by : [Function Call](#_db3e44e523a232e5b77a133d74842e81) [\*] *Redefines*: categorizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)



Function calls using a function declaration.

implemented by : [Expression](#_f9bba899ada544a47c36bb071e9024f5) [0..1] *Subsets*: contextualizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)



Expression which defined the implementation of a function.

### Association Function Type

**package** SIMF Conceptual Model::Expressions

#### Association Ends

calls : [Function Type](#_cff99d2f22ee84a9e95ea582786a897b) [1] *Subsets*: contextualizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)



Function called

is used by : [Function Call](#_db3e44e523a232e5b77a133d74842e81) [\*] *Subsets*: contextualizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)



Function calls using a function declaration.

### Class Object Operation Type

An operation bound to a spcific "reciever" in the OO sense.

#### Direct Supertypes

[Function Type](#_cff99d2f22ee84a9e95ea582786a897b)

**package** SIMF Conceptual Model::Expressions

#### Associations

reciever : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1]



The property that is the receiver of an object operation. The receiver must be a property of the function.

### Association OO Target

**package** SIMF Conceptual Model::Expressions

#### Association Ends

reciever : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1]



The property that is the receiver of an object operation. The receiver must be a property of the function.

recieved by : [Object Operation Type](#_e6c2e5d52e1652a6c3d27d411345c754) [\*]



### Association Return type

**package** SIMF Conceptual Model::Expressions

#### Association Ends

resulting type : [Type](#_dfe1514224ca21cedba7b2b29802db50) [\*]



Type of the result of a function

returned by : [Expression Context](#_d847ee03faa23264a18dd452d21972fc) [\*]



Ty

### Class Traversal

Traversal from the current context to another across a relation/situation.   
  
A traversal is a situation such that the situations bindings may hold other properties of a traversal constant where "traverse through" is the dependent variable. The traversal shall be considered to have the type of the relation it is traversing. Traversing binary relations does not require any bindings.

#### Direct Supertypes

[Expression](#_f9bba899ada544a47c36bb071e9024f5), [Structure](#_8c517cf1950741c0f89edebf828214cc)

**package** SIMF Conceptual Model::Expressions

#### Attributes

traverse to relation : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f) [1] = false



Where traverse to relation is false, the traversal will return the bound element(s) of the property from the current context via any intermediate relationships.  
  
Where traverse to relation is true, the traversal shall return the situation/relationship owning the property binding.  
  
By default, traverse to relation is false.

inverse : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f) [1] = false



Indicates that the traversal is defined based on properties that reference the current context. This results in traversing "backwards" across a property to an inverse property or the relation.  
Where the situation referencing the property has more than one other property, <traverse to relation> must be TRUE.

#### Associations

traverses through : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1]



Traversals using a property

### Association traverse through

Property of current context which will be traversed.

**package** SIMF Conceptual Model::Expressions

#### Association Ends

traverses through : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1]



Traversals using a property

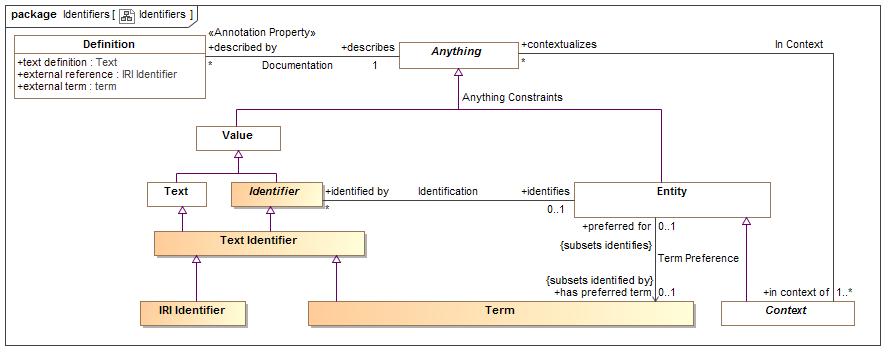
traversed by : [Traversal](#_0492440b12b90a76377a15324efa2182) [\*]



## SIMF Conceptual Model::Identifiers

Terms and identifiers provide for signs for (ways to identify) anything.

### Diagram: Identifiers



1. Identifiers

### Association Identification

**package** SIMF Conceptual Model::Identifiers

#### Association Ends

identifies : [Entity](#_eb8398b5a178c638b98597120ec51c4d) [0..1]



A named concept

identified by : [Identifier](#_18f8ef1b23e6cdf9278bd94f24f73c26) [\*]



Names for a concept

### Class Identifier

Anything that is used to identify a concept

#### Direct Supertypes

[Value](#_a739673c8d53da123e392b7e5059ceec)

**package** SIMF Conceptual Model::Identifiers

#### Associations

identifies : [Entity](#_eb8398b5a178c638b98597120ec51c4d) [0..1]



A named concept

### Class IRI Identifier

A IRI Identifier for a concept,   
as defined in [RFC3987].  
Includes URIs.

#### Direct Supertypes

[Text Identifier](#_c9d4914a019b89a37f1f18103ebaf817)

**package** SIMF Conceptual Model::Identifiers

### Class Term

A term / element name: the binding between a label and a named concept.

#### Direct Supertypes

[Text Identifier](#_c9d4914a019b89a37f1f18103ebaf817)

**package** SIMF Conceptual Model::Identifiers

### Class Text Identifier

#### Direct Supertypes

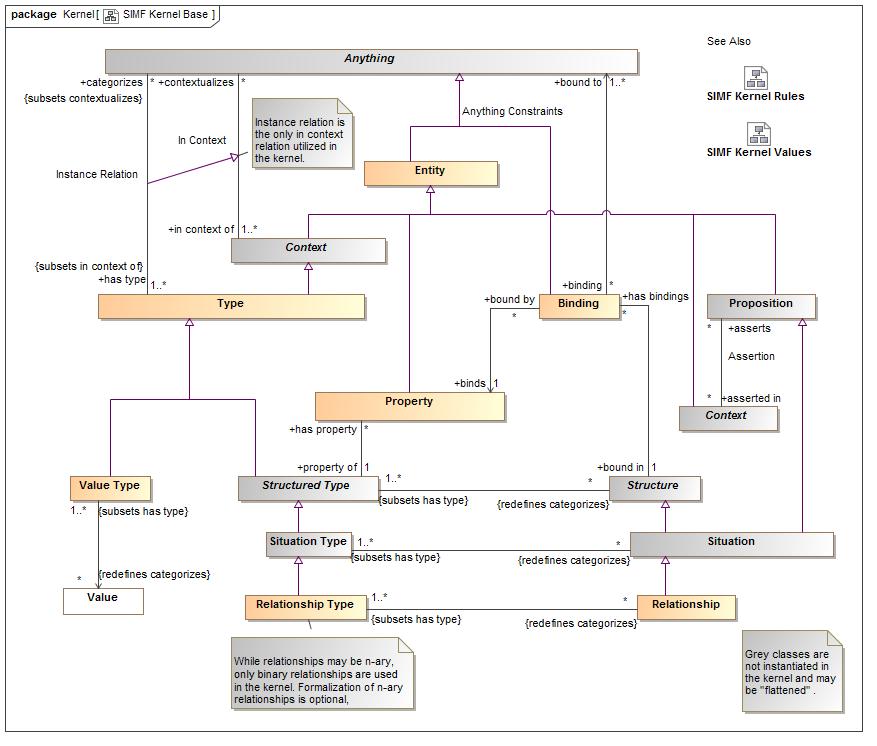
[Identifier](#_18f8ef1b23e6cdf9278bd94f24f73c26), [Text](#_0e6e6fe0a29fb43221940aa4118b04a2)

**package** SIMF Conceptual Model::Identifiers

## SIMF Conceptual Model::Kernel

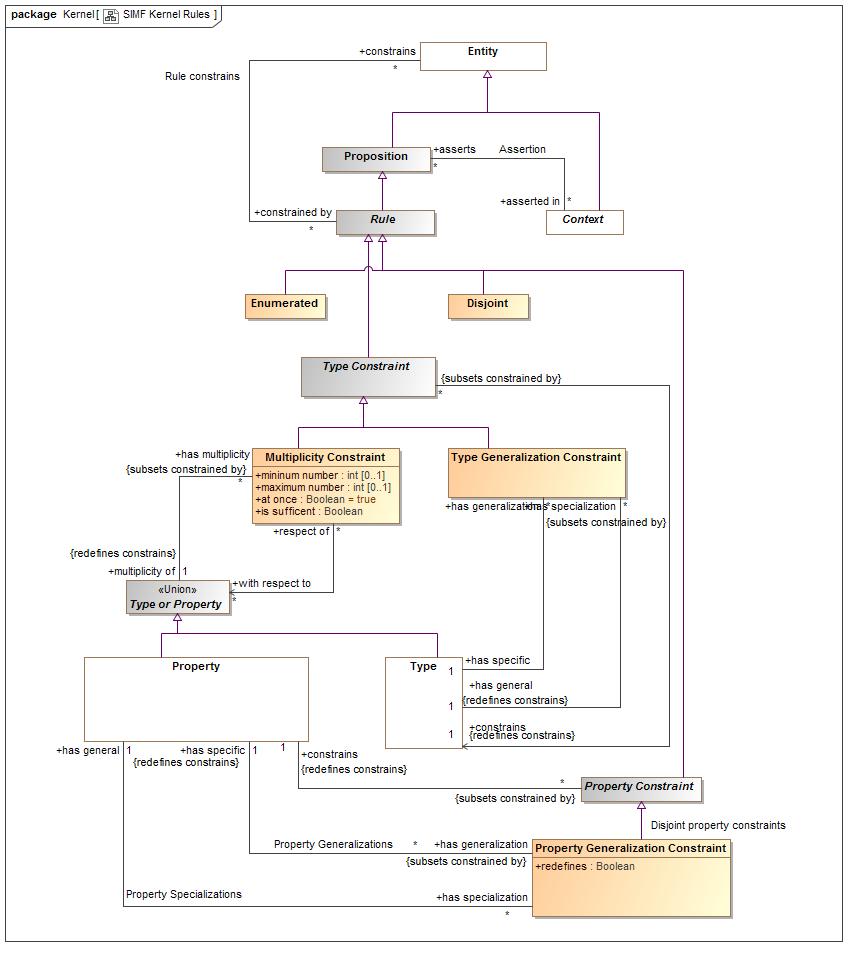
The kernel defines concrete classes that are used to define the SIMF language

### Diagram: SIMF Kernel Base



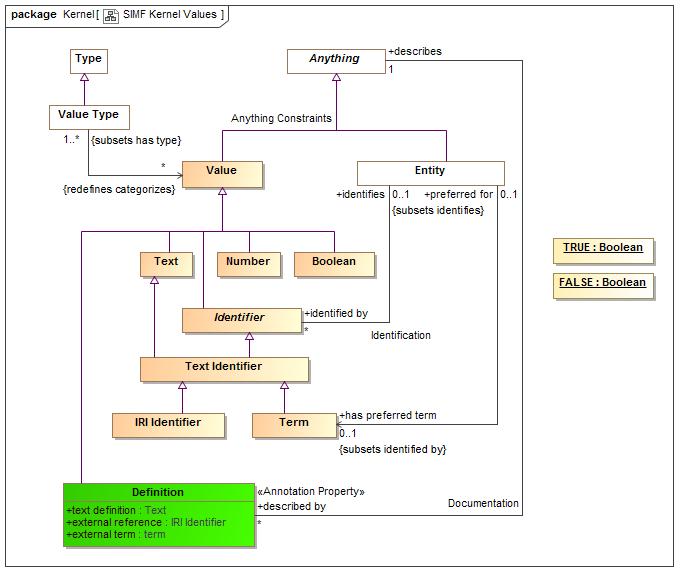
1. SIMF Kernel Base

### Diagram: SIMF Kernel Rules



1. SIMF Kernel Rules

### Diagram: SIMF Kernel Values

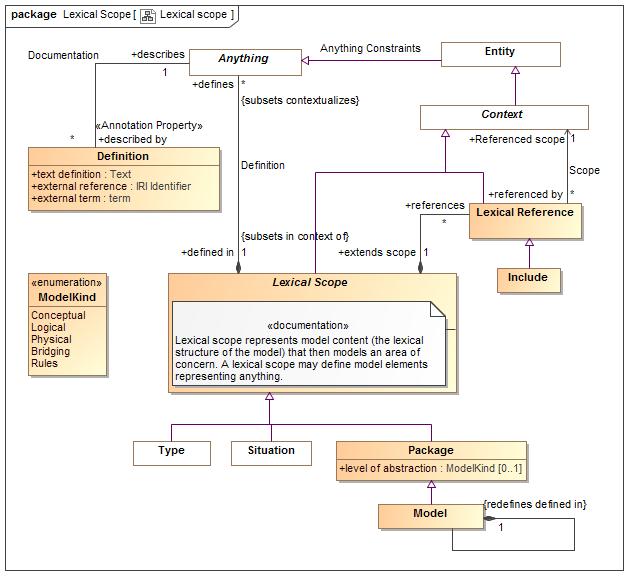


1. SIMF Kernel Values

## SIMF Conceptual Model::Lexical Scope

Lexical scope defines the structure of models

### Diagram: Lexical scope



1. Lexical scope

An indicator that the referenced context is asserted by (included by reference) the referencing context.

### Association Definition

**package** SIMF Conceptual Model::Lexical Scope

#### Association Ends

defines : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [\*]



Model elements defined within a lexical scope.

defined in : [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151) [1]



Lexical scope defining model elements.

### Class Definition

Natural language definition of a concept.   
May be in the context of a natural language.

#### Direct Supertypes

[Value](#_a739673c8d53da123e392b7e5059ceec)

**package** SIMF Conceptual Model::Lexical Scope

#### Attributes

text definition : [Text](#_0e6e6fe0a29fb43221940aa4118b04a2)



Text describing a concept in natural language

external reference : [IRI Identifier](#_f904ff1da5bfc3387d892b7e0fe9ecb1)



reference to an external information resource that further defines the element.

external term : [term](#_b8c41a054f53ee779611ae923ea38f47)



Specific term in an external resource that further defined an element.

#### Associations

describes : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [1]



Described element

### Association Documentation

**package** SIMF Conceptual Model::Lexical Scope

#### Association Ends

describes : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [1]



Described element

described by : [Definition](#_1a6d88e097d757268d09f68af82fbd34) [\*]



Informal descriptions

### Class Include

An external scope that is visible and asserted by the owning lexical scope.

#### Direct Supertypes

[Lexical Reference](#_0315319befc74caa0a2a7d36cff333c0)

**package** SIMF Conceptual Model::Lexical Scope

### Class Lexical Reference

An external scope that is visible to but not necessarily asserted by the owning lexical scope.

#### Direct Supertypes

[Context](#_66d62b068053cee3464e1e03e6035eed)

**package** SIMF Conceptual Model::Lexical Scope

#### Associations

Referenced scope : [Context](#_66d62b068053cee3464e1e03e6035eed) [1]



A referenced context, potentially in another model, that provides visibility to the elements in that context.

extends scope : [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151) [1]



Scope which is referenced and used by a lexical scope.

### Class Lexical Scope

Lexical scope represents model content (the lexical structure of the model) that then models an area of concern. A lexical scope may define model elements representing anything.

#### Direct Supertypes

[Context](#_66d62b068053cee3464e1e03e6035eed)

**package** SIMF Conceptual Model::Lexical Scope

#### Associations

defines : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [\*] *Subsets*: contextualizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)



Model elements defined within a lexical scope.

references : [Lexical Reference](#_0315319befc74caa0a2a7d36cff333c0) [\*]



The context that is referenced by a given perspective.

### Class Model

A root package. A model has no owner and may be directly referenced as an independent information resource. A model is defined in it's self.

#### Direct Supertypes

[Package](#_0506f167988dfda7ae188b66aefe4f05)

**package** SIMF Conceptual Model::Lexical Scope

#### Associations

: [Lexical Reference](#_0315319befc74caa0a2a7d36cff333c0) [\*]



: [Lexical Reference](#_0315319befc74caa0a2a7d36cff333c0) [\*]



### Class Package

A group of model elements that provides a definitional scope for those elements.

#### Direct Supertypes

[Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151)

**package** SIMF Conceptual Model::Lexical Scope

#### Attributes

level of abstraction : [ModelKind](#_f8d5fc9c395254768dfd7ddce8807215) [0..1]



Level of abstraction a package (or model) represents.

### Association Reference

**package** SIMF Conceptual Model::Lexical Scope

#### Association Ends

references : [Lexical Reference](#_0315319befc74caa0a2a7d36cff333c0) [\*]



The context that is referenced by a given perspective.

extends scope : [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151) [1]



Scope which is referenced and used by a lexical scope.

### Association Scope

**package** SIMF Conceptual Model::Lexical Scope

#### Association Ends

Referenced scope : [Context](#_66d62b068053cee3464e1e03e6035eed) [1]



A referenced context, potentially in another model, that provides visibility to the elements in that context.

referenced by : [Lexical Reference](#_0315319befc74caa0a2a7d36cff333c0) [\*]



### Class term

**package** SIMF Conceptual Model::Lexical Scope

#### Enumeration ModelKind

package SIMF Conceptual Model::Lexical Scope

public enum ModelKind

{Conceptual, Logical, Physical, Bridging, Rules}

##### Literals

Conceptual



Logical



Physical



Bridging



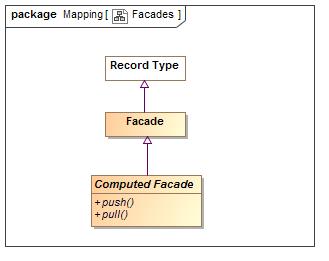
Rules



## SIMF Conceptual Model::Mapping

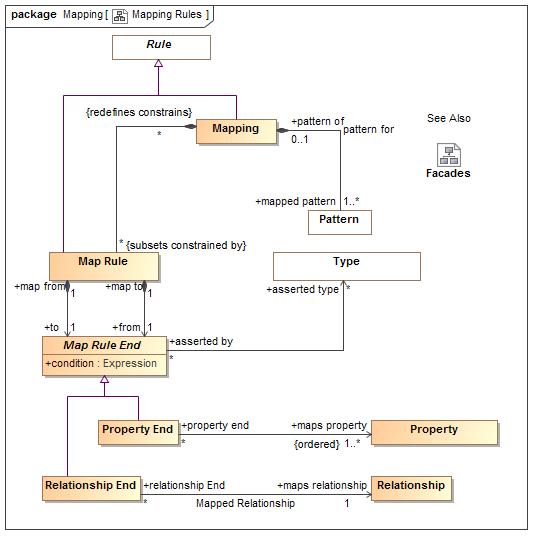
Mapping rules define how data represents concepts.

### Diagram: Facades



1. Facades

### Diagram: Mapping Rules



1. Mapping Rules

### Class Computed Facade

A facade that is computed by calling external methods.

#### Direct Supertypes

[Facade](#_90ac762f4f29c31e6c33236231df6a9a)

**package** SIMF Conceptual Model::Mapping

#### Operations

public push ()



An operation called to evoke the behavior associated with a new facade element being created or modified.

public pull ()



An operation called to evoke the behavior associated with a facade representing existing elements.

### Class Facade

An intermediary data type used to hold common mappings. Facades may be computed and/or have mapping rules.

#### Direct Supertypes

[Record Type](#_d2ebf1b96697234b6aef9b3bfac15784)

**package** SIMF Conceptual Model::Mapping

### Association from rue

**package** SIMF Conceptual Model::Mapping

#### Association Ends

from : [Map Rule End](#_0d8a19bfafdae6e590a12e54ebcff122) [1]



One end of a mapping, to be used for more concrete side.

map to : [Map Rule](#_63d69e49de8214503f0947e7f9dbc652) [1]



### Class Map Rule

A statement that the 2 ends represent the same things or information about a thing.

#### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

**package** SIMF Conceptual Model::Mapping

#### Associations

from : [Map Rule End](#_0d8a19bfafdae6e590a12e54ebcff122) [1]



One end of a mapping, to be used for more concrete side.

to : [Map Rule End](#_0d8a19bfafdae6e590a12e54ebcff122) [1]



One end of a mapping, to be used for more abstract end.

: [Mapping](#_551417ad3c6e740d8b880bee8085a718) [\*] *Redefines*: constrains:[Entity](#_eb8398b5a178c638b98597120ec51c4d)



### Class Map Rule End

One end of a mapping from one thing to another that may be qualified with a condition.  
The set of elements to be mapped is the union of the sets off all mapped types and mapped variables that conform to the condition.  
Mapping rules are constrained to apply to only conforming types or types that represent the mapped ends.

**package** SIMF Conceptual Model::Mapping

#### Attributes

condition : [Expression](#_f9bba899ada544a47c36bb071e9024f5)



Condition that must be TRUE for the mapping rule to apply.

#### Associations

asserted type : [Type](#_dfe1514224ca21cedba7b2b29802db50) [\*]



Additional type that will be asserted for the end. e.g. a unit type.

### Association Map Rule Type Assertion

**package** SIMF Conceptual Model::Mapping

#### Association Ends

asserted type : [Type](#_dfe1514224ca21cedba7b2b29802db50) [\*]



Additional type that will be asserted for the end. e.g. a unit type.

asserted by : [Map Rule End](#_0d8a19bfafdae6e590a12e54ebcff122) [\*]



### Association Mapped Property

Type that is the source or target of the mapping

**package** SIMF Conceptual Model::Mapping

#### Association Ends

maps property : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1..\*]



Type that defines a set of elements to map to the other side of the mapping rule based on a property. Where more than one property is specified a property path is followed.

property end : [Property End](#_7decbae485fa9be88d63fca6f68547ec) [\*]



### Association Mapped Relationship

**package** SIMF Conceptual Model::Mapping

#### Association Ends

maps relationship : [Relationship](#_f7a7f80baaeb7cc3f36c45e96eacd166) [1]



Type that defines a set of elements to map to the other side of the mapping rule based on a relationship contained in the pattern.

relationship End : [Relationship End](#_2d6f605b4614a2c3324d5d7f1fe4743a) [\*]



### Class Mapping

A mapping is a rule that defines how different representations of the same things correspond.  
Patterns define a set of related elements to be mapped.   
Types may be defined to be a representation (data about) a concept.  
Map rules define how elements in each of the patterns are mapped, bidirectionally.

#### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

**package** SIMF Conceptual Model::Mapping

#### Associations

: [Map Rule](#_63d69e49de8214503f0947e7f9dbc652) [\*] *Subsets*: constrained by:[Rule](#_82919e40af9ad2e13647e9d37bbf0956)



mapped pattern : [Pattern](#_8d9c945b6f864c34fdd7a91d4d62755f) [1..\*]



A pattern of elements to be mapped. Where divergent models are mapped, each would be defined in its own pattern.

### Association pattern for

**package** SIMF Conceptual Model::Mapping

#### Association Ends

mapped pattern : [Pattern](#_8d9c945b6f864c34fdd7a91d4d62755f) [1..\*]



A pattern of elements to be mapped. Where divergent models are mapped, each would be defined in its own pattern.

pattern of : [Mapping](#_551417ad3c6e740d8b880bee8085a718) [0..1]



Mapping which defines a pattern

### Class Property End

An end of a map rule where the end maps a property, the elements bound to that property for each match of the pattern will be mapped.  
If more than one property is specified a path through such properties will be defined.  
The first property must be a property of one of the patterns of the mapping.

#### Direct Supertypes

[Map Rule End](#_0d8a19bfafdae6e590a12e54ebcff122)

**package** SIMF Conceptual Model::Mapping

#### Associations

maps property : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1..\*]



Type that defines a set of elements to map to the other side of the mapping rule based on a property. Where more than one property is specified a property path is followed.

### Class Relationship End

An end of a map rule where the end maps to a relationship, the instances of the relationship between the properties will be mapped.  
The relationship must be defined in one of the mapped patterns.

#### Direct Supertypes

[Map Rule End](#_0d8a19bfafdae6e590a12e54ebcff122)

**package** SIMF Conceptual Model::Mapping

#### Associations

maps relationship : [Relationship](#_f7a7f80baaeb7cc3f36c45e96eacd166) [1]



Type that defines a set of elements to map to the other side of the mapping rule based on a relationship contained in the pattern.

### Association to rule

**package** SIMF Conceptual Model::Mapping

#### Association Ends

to : [Map Rule End](#_0d8a19bfafdae6e590a12e54ebcff122) [1]



One end of a mapping, to be used for more abstract end.

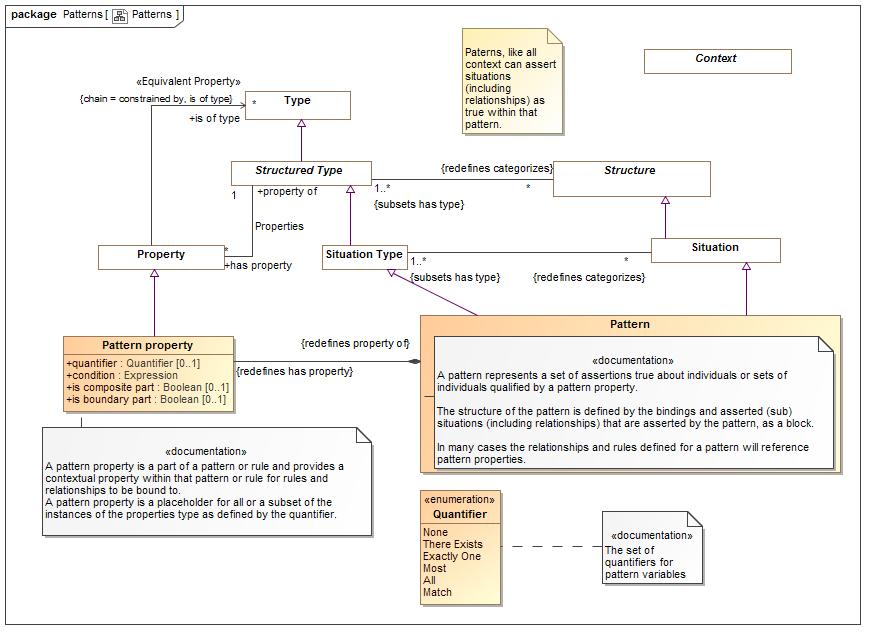
map from : [Map Rule](#_63d69e49de8214503f0947e7f9dbc652) [1]



## SIMF Conceptual Model::Patterns

Templates are patterns that may then be expressed as instances of the pattern.

### Diagram: Patterns



1. Patterns

### Class Pattern

A pattern represents a set of assertions true about individuals or sets of individuals qualified by a pattern property.   
   
The structure of the pattern is defined by the bindings and asserted (sub) situations (including relationships) that are asserted by the pattern, as a block.  
  
In many cases the relationships and rules defined for a pattern will reference pattern properties.

#### Direct Supertypes

[Situation](#_bed9725f6bf8a4fff6fd1087850b7259), [Situation Type](#_c568596a9f8653e157460ae519e96906)

**package** SIMF Conceptual Model::Patterns

#### Associations

pattern of : [Mapping](#_551417ad3c6e740d8b880bee8085a718) [0..1]



Mapping which defines a pattern

: [Pattern property](#_4d83e476040c7444758dda440d3096fc) *Redefines*: has property:[Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d)



### Class Pattern property

A pattern property is a part of a pattern or rule and provides a contextual property within that pattern or rule for rules and relationships to be bound to.  
A pattern property is a placeholder for all or a subset of the instances of the properties type as defined by the quantifier.

#### Direct Supertypes

[Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d)

**package** SIMF Conceptual Model::Patterns

#### Attributes

quantifier : [Quantifier](#_b82d09af5a9584abc6560d3bfb03e524) [0..1]



An assertion that defines a quantification (based on a subtype) over a context. A concept is considered the quantified property.  
e.g. for all people p: People is the context and P is the quantified property. In SIMF the quantified property would typically be named <quantifier> <type>. So the above quantified property would be named "all people". The quantified property will be asserted to have the quantified type.  
  
A Quantifier would be "mixed in" with a concrete subtype of Role.

condition : [Expression](#_f9bba899ada544a47c36bb071e9024f5)



Condition that must be true for all individuals bound to a property.

is composite part : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f) [0..1]



True if the property represents an internal part of the pattern. False is the property is external to the pattern.

is boundary part : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f) [0..1]



True if the part is on the boundary of the pattern and connectible (may have relationships) externally.

#### Associations

: [Pattern](#_8d9c945b6f864c34fdd7a91d4d62755f) *Redefines*: property of:[Structured Type](#_50241f5936e61055293ca95f860768d8)



#### Enumeration Quantifier

The set of quantifiers for pattern variables

package SIMF Conceptual Model::Patterns

public enum Quantifier

{None, There Exists, Exactly One, Most, All, Match}

##### Literals

None



A quantifier where no instance of the type fills the role.

There Exists



The existential quantifier - at least one. A logical "supertype" of "One of" and "Most"

Exactly One



The existential quantifier limited to exactly one of a potentially larger set

Most



A stratified existential quantifier with a default for a "typical" value - example: People <typically> have 2 arms.

All



The universal quantifier - the quantified property is a stand-in for all elements of the existent of the quantified type

Match

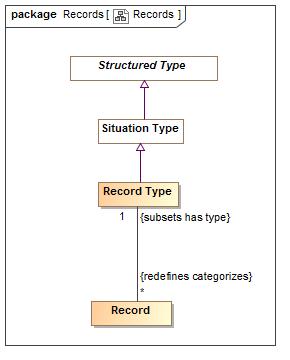


Select is used in query and mapping patterns, all elements of the classified type that match the pattern are selected as instances of the pattern.

## SIMF Conceptual Model::Records

Records represent structured data, usually used for logical or physical data models, not conceptual models.

### Diagram: Records



1. Records

### Class Record

A record of the condition of a situation at a point in time - this includes facts, speech acts and DBMS records.   
Records are typically used in data representations, not conceptual models.

#### Direct Supertypes

[Situation](#_bed9725f6bf8a4fff6fd1087850b7259)

**package** SIMF Conceptual Model::Records

#### Associations

: [Record Type](#_d2ebf1b96697234b6aef9b3bfac15784) [1] *Subsets*: has type:[Type](#_dfe1514224ca21cedba7b2b29802db50)



### Class Record Type

Type of the record of the condition of a situation at a point in time - this includes facts, speech acts and DBMS records.   
A record type may involve variant and invariant types as variables. Those that are enumerated in a "uniqueness constraint" are invariant (independent variables) uniquely identify the situation which is the subject of the fact type where as the other variables may change over time (dependent variables).  
Record types may be grounded in atomic relations by using invariant conditions.

#### Direct Supertypes

[Situation Type](#_c568596a9f8653e157460ae519e96906)

**package** SIMF Conceptual Model::Records

#### Associations

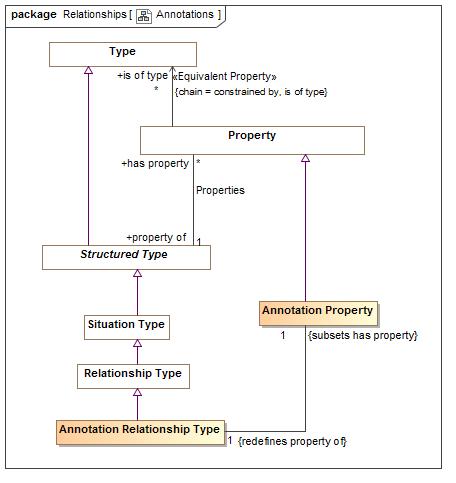
: [Record](#_8b38efa9c56da3bc8ecb501e56419e41) [\*] *Redefines*: categorizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)



## SIMF Conceptual Model::Relationships

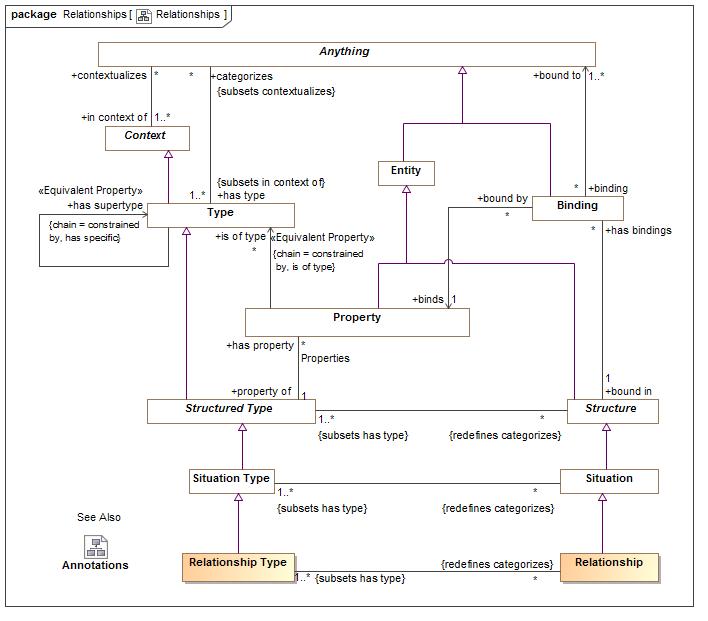
Relations are primitive facts about anything, relating at least 2 individuals through the roles they play.

### Diagram: Annotations



1. Annotations

### Diagram: Relationships



1. Relationships

Relations are atomic situations that bind 2 or more roles as a fact.

### Class Annotation Property

An annotation property is a specialization of property where the referenced elements represent metadata about the related proposition, structure or information (or model element) rather than a fact or condition of the domain being represented.  
For an annotation property, <is of type> describes instances of the structured type for which the property is defined.  
Typical uses of annotations include provenance of information, when a record was created, etc.

#### Direct Supertypes

[Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d)

**package** SIMF Conceptual Model::Relationships

#### Associations

: [Annotation Relationship Type](#_7a2b4b15c2377efe429d3d9e5b30b859) [1] *Redefines*: property of:[Structured Type](#_50241f5936e61055293ca95f860768d8)



### Class Annotation Relationship Type

An annotation relationship type is a specialization of relationship type where the relationship represents metadata about the related element, structure or information (or model element) rather than a fact or condition of the domain being represented.  
  
Typical uses of annotations include provenance of information, when a record was created, etc.

#### Direct Supertypes

[Relationship Type](#_adbc34bb07fb06a9fcdba6f84fc4c37a)

**package** SIMF Conceptual Model::Relationships

#### Associations

: [Annotation Property](#_630a40beaf4bbbac635cfcacaaf353d1) [1] *Subsets*: has property:[Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d)



### Class Relationship

Any proposition stated within a context. Each proposition has <Binding>s to the variables of the relationship type that uniquely identify the relationship instance and do not change.

#### Direct Supertypes

[Situation](#_bed9725f6bf8a4fff6fd1087850b7259)

**package** SIMF Conceptual Model::Relationships

#### Associations

: [Relationship Type](#_adbc34bb07fb06a9fcdba6f84fc4c37a) [1..\*] *Subsets*: has type:[Type](#_dfe1514224ca21cedba7b2b29802db50)



### Class Relationship Type

Any type of proposition or fact stated within a context that is true or false within that context. Each proposition has a number of properties that uniquely identify the relationship and do not change.   
  
A relationship may be true of false within its context but is atomic in its truth value.  
Relationships may participate in (be bound to) other relationships and such bindings may change over time.  
  
Note that this differentiates properties of a relationship which define its identity in terms of the related individuals from other relationships that reference it, such other relationships have their own identity, truth value and life cycle.  
  
The properties of a relationship represent the verb phrases connecting the relationship with the related types.

#### Direct Supertypes

[Situation Type](#_c568596a9f8653e157460ae519e96906)

**package** SIMF Conceptual Model::Relationships

#### Associations

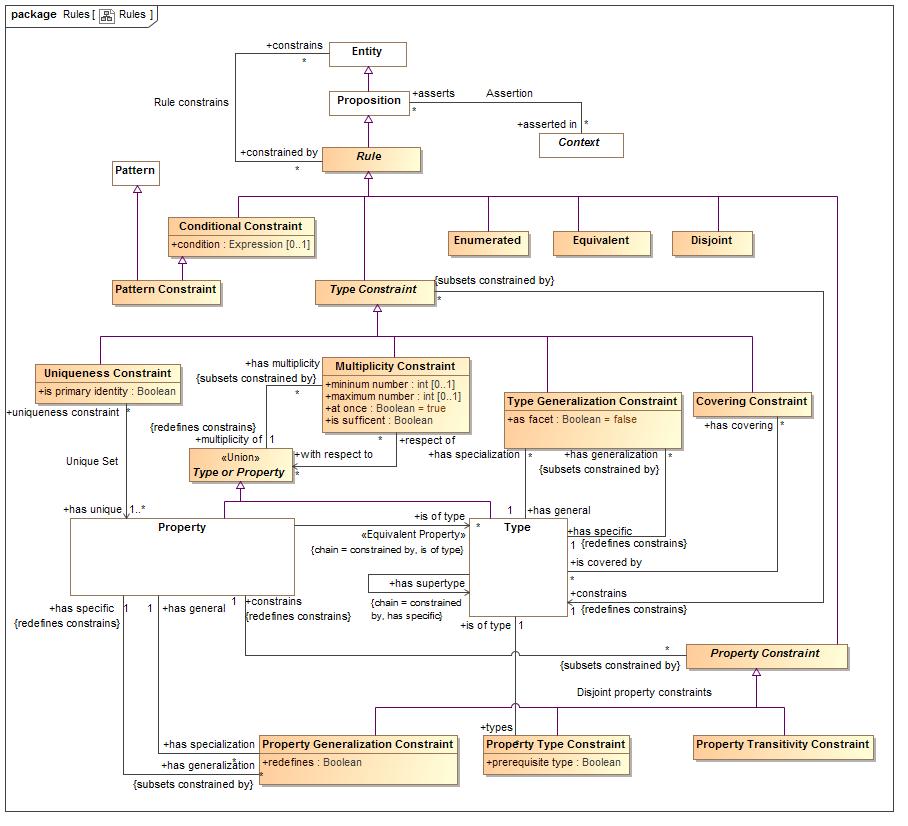
: [Relationship](#_f7a7f80baaeb7cc3f36c45e96eacd166) [\*] *Redefines*: categorizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)



## SIMF Conceptual Model::Rules

Rule define constraints or behaviors that must be applied in specified context.

### Diagram: Rules



1. Rules

### Class Conditional Constraint

A rule with a general expression as a condition

#### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

**package** SIMF Conceptual Model::Rules

#### Attributes

condition : [Expression](#_f9bba899ada544a47c36bb071e9024f5) [0..1]



Condition that must be TRUE for the rule to "fire". All other values are FALSE.

### Class Covering Constraint

A constraint that the extent (categorized things) of the <constrains> type is equivalent to the union of the extents of the <is covered by> types.

#### Direct Supertypes

[Type Constraint](#_ded47679f07683882f8f128d6911711a)

**package** SIMF Conceptual Model::Rules

#### Associations

is covered by : [Type](#_dfe1514224ca21cedba7b2b29802db50) [\*]



the union of the set of types referenced by <is covered by> defines the extent of the type the covering constraint <constrains>.  
  
The <constrains> type must be a direct supertype of all <is covered by> types.

### Class Disjoint

Rule that different elements defined in models do not and may not denote any of the same set of real or possible world entities.  
When applied to a context (including types) all elements contextualized are included in the set of disjoint individuals.  
Each context the rule <constrains> is disjoint.

#### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

**package** SIMF Conceptual Model::Rules

### Class Enumerated

The contextualized elements of the <constrains> context is a closed (enumerated) set, it can not be extended. A.K.A. "Closed World Assumption". Elements may not be owned by any context other than the one specified.

#### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

**package** SIMF Conceptual Model::Rules

### Class Equivalent

Rule that different elements defined in models denote the same set of real or possible world entities. Each context the rule <constrains> is equivalent - has the same extent.

#### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

**package** SIMF Conceptual Model::Rules

### Association Generalizations

**package** SIMF Conceptual Model::Rules

#### Association Ends

has general : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1]



The general type in the Generalization rule.

has specialization : [Type Generalization Constraint](#_5f3998cf1a072f724861db93cee66cbf) [\*]



Generalization/Specialization rules for a type.

### Class Multiplicity Constraint

Constrains the number of times members <multiplicity of> may be involved in situations that bind the constraining type or property. This provides for complex multiplicities across n-ary situations and relationships.   
The constraint is defined within (applies to) types, which includes relationships.  
The number of instances binding to the property for the set of instances bound to <with respect to> shall be limited by the minimum and maximum number of the multiplicity.

#### Direct Supertypes

[Type Constraint](#_ded47679f07683882f8f128d6911711a)

**package** SIMF Conceptual Model::Rules

#### Attributes

mininum number : [int](#_0d30278207cac92be6fa561506a22f92) [0..1]



Minimum number in a set as constrained by a multiplicity.

maximum number : [int](#_0d30278207cac92be6fa561506a22f92) [0..1]



Maximum number in a set as constrained by a multiplicity.

at once : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f) = true



When at once is true, the constraint applies for each snapshot in time but not across snapshots (e.g. a car can have at most one driver at a time). When at once is false the constraint applies across all time (e.g. a person has exactly one birth mother across all time).

is sufficent : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)



One of the set of sufficient conditions that will infer the type designated in <constrains>.

#### Associations

multiplicity of : [Type or Property](#_0ae2de066cc60f0d0ea185e32c2ee88c) [1] *Redefines*: constrains:[Entity](#_eb8398b5a178c638b98597120ec51c4d)



The type or property that is the subject of the constraint.

with respect to : [Type or Property](#_0ae2de066cc60f0d0ea185e32c2ee88c) [\*]



One or more types or variables that define the <from> side of a multiplicity.   
  
Where with respect to is undefined and <has multiplicity> is a property, all properties that are <property of> the same structured type as <multiplicity of> shall be considered the set of <with respect to> properties. I.e. all the "other ends" of a relationship.

### Association Multiplicity Perspective

**package** SIMF Conceptual Model::Rules

#### Association Ends

with respect to : [Type or Property](#_0ae2de066cc60f0d0ea185e32c2ee88c) [\*]



One or more types or variables that define the <from> side of a multiplicity.   
  
Where with respect to is undefined and <has multiplicity> is a property, all properties that are <property of> the same structured type as <multiplicity of> shall be considered the set of <with respect to> properties. I.e. all the "other ends" of a relationship.

respect of : [Multiplicity Constraint](#_4ff1432ad36ac8beb6cbb7e9323d9f24) [\*]



### Association Multiplicity Target

Type with references to the "to type", normally a role or a situation type - but can be any type.

**package** SIMF Conceptual Model::Rules

#### Association Ends

multiplicity of : [Type or Property](#_0ae2de066cc60f0d0ea185e32c2ee88c) [1]



The type or property that is the subject of the constraint.

has multiplicity : [Multiplicity Constraint](#_4ff1432ad36ac8beb6cbb7e9323d9f24) [\*]



Multiplicity constraint of a type or property in a context.

### Class Pattern Constraint

A pattern rule asserts that the assertions it contains are true for the pattern variables identified.

#### Direct Supertypes

[Conditional Constraint](#_3d425949001fb1cb0502a6157c8cf51e), [Pattern](#_8d9c945b6f864c34fdd7a91d4d62755f)

**package** SIMF Conceptual Model::Rules

### Class Property Constraint

Constraints that apply to properties.

#### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

**package** SIMF Conceptual Model::Rules

#### Associations

constrains : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1] *Redefines*: constrains:[Entity](#_eb8398b5a178c638b98597120ec51c4d)



### Class Property Generalization Constraint

Property may be marked as the specialization of another. The specialized property represents a set that is the same as or a subset of the more general property.  
  
Where <redefines> is true the more specific property subsumes the more general property in the definition context. In this case the more general and more specific sets are equivalent.   
  
A more specific property shall be consistent with the property it specializes, but it may add specific constraints or other details that are particular to instances of the specializing situation that do not contradict constraints in the general context.

#### Direct Supertypes

[Property Constraint](#_91be190e8014514a597300b286148d3e)

**package** SIMF Conceptual Model::Rules

#### Attributes

redefines : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)



Defines the generalization as a redefinition, subsuming the more general property.  
  
Where <redefines> is true the more specific property subsumes the more general property in the definition context. In this case the more general and more specific sets are equivalent. A property may be redefined multiple times, as long as it is unambiguous which definition applies for a particular instance.  
  
Where <redefines> is false or not defined the more specific property represents a subset of the more general property.

#### Associations

has general : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1]



THe more general property as part of a property generalization

has specific : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1] *Redefines*: constrains:[Entity](#_eb8398b5a178c638b98597120ec51c4d)



The more specific property as part of a property generalization

### Association Property Generalizations

**package** SIMF Conceptual Model::Rules

#### Association Ends

has specific : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1] *Redefines*: constrains: [Entity](#_eb8398b5a178c638b98597120ec51c4d)



The more specific property as part of a property generalization

has generalization : [Property Generalization Constraint](#_8453c07e3c7aaeab8e715f3569b782ab) [\*] *Redefines*: constrains: [Entity](#_eb8398b5a178c638b98597120ec51c4d)



Generalization/Specialization rules for a property.

### Association Property Specializations

**package** SIMF Conceptual Model::Rules

#### Association Ends

has general : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1] *Redefines*: constrains: [Entity](#_eb8398b5a178c638b98597120ec51c4d)



THe more general property as part of a property generalization

has specialization : [Property Generalization Constraint](#_8453c07e3c7aaeab8e715f3569b782ab) [\*] *Redefines*: constrains: [Entity](#_eb8398b5a178c638b98597120ec51c4d)



Generalizations of a property

### Class Property Transitivity Constraint

A transitive property interlinks two individuals A and C whenever it interlinks A with B and B with C for some individual B.   
For example "larger than" is transitive in that if Joe is larger than Sue and Sue is Larger then Sam, then Joe is larger than Sam.

#### Direct Supertypes

[Property Constraint](#_91be190e8014514a597300b286148d3e)

**package** SIMF Conceptual Model::Rules

### Association Property Type

**package** SIMF Conceptual Model::Rules

#### Association Ends

is of type : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1] *Redefines*: constrains: [Entity](#_eb8398b5a178c638b98597120ec51c4d)



The required type of a property.

types : [Property Type Constraint](#_3e4d7d36a3b2cb4c9b85c1b88930178a) [\*] *Redefines*: constrains: [Entity](#_eb8398b5a178c638b98597120ec51c4d)



Properties typed by a type

### Class Property Type Constraint

A property type rule defines the type(s) of a property in a given context.  
All elements bound to a property must have the type <has type>. Has type may be pre-existing or inferred based on the value of <prerequisite type>.

#### Direct Supertypes

[Property Constraint](#_91be190e8014514a597300b286148d3e)

**package** SIMF Conceptual Model::Rules

#### Attributes

prerequisite type : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)



If true, <has type> is a prerequisite - the bound element must be of the given type for the property to be bound. A non prerequisite type will cause a binding to infer <has type>, provided all prerequisite types have been satisfied.

#### Associations

is of type : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1]



The required type of a property.

### Class Rule

A rule is a proposition that constrains one or more entities by limiting possible conditions or producing some effect.

#### Direct Supertypes

[Proposition](#_3bd7c7d249201ad6f2447c6d182ba7f1)

**package** SIMF Conceptual Model::Rules

#### Associations

constrains : [Entity](#_eb8398b5a178c638b98597120ec51c4d) [\*]



The entity or entities constrained by a rule.   
Where a rule constrains a context, all things contextualized by the context shall be subject to the rule.

### Association Rule constrains

**package** SIMF Conceptual Model::Rules

#### Association Ends

constrains : [Entity](#_eb8398b5a178c638b98597120ec51c4d) [\*]



The entity or entities constrained by a rule.   
Where a rule constrains a context, all things contextualized by the context shall be subject to the rule.

constrained by : [Rule](#_82919e40af9ad2e13647e9d37bbf0956) [\*]



Rules applying to things contextualized by a context.

### Association Specializations

**package** SIMF Conceptual Model::Rules

#### Association Ends

has specific : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1]



The specific type in a generalization rule.

has generalization : [Type Generalization Constraint](#_5f3998cf1a072f724861db93cee66cbf) [\*]



Generalizations of a type

### Class Type Constraint

Constraints applicable within any situation type, including Relationships.  
Type constraints are not necessarily defined in the same context as the type. Other context may assert constraints on any type.

#### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

**package** SIMF Conceptual Model::Rules

#### Associations

constrains : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1] *Redefines*: constrains:[Entity](#_eb8398b5a178c638b98597120ec51c4d)



### Class Type Generalization Constraint

A Specialization is a taxonomic relationship between a more general <generalizes> type and a more specific <specializes> type. Each instance of the specific type is also an instance of the general type. The specific Classifier inherits the properties and rules of the more general type.

#### Direct Supertypes

[Type Constraint](#_ded47679f07683882f8f128d6911711a)

**package** SIMF Conceptual Model::Rules

#### Attributes

as facet : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f) = false



If <as facet> is true the specialized type is "non rigid" with respect to the general type - that is the <specializes> type may change over the lifetime of instances of the base type. The <specializes> type will be inferred to be a Facet. e.g. "Registered voter".   
  
If <as facet> is false the specialized type is "rigid" with respect to the general type - that is the <specializes> type may not change over the lifetime of instances of the base type. e.g. "Tree".

#### Associations

has specific : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1] *Redefines*: constrains:[Entity](#_eb8398b5a178c638b98597120ec51c4d)



The specific type in a generalization rule.

has general : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1]



The general type in the Generalization rule.

### Class Type or Property

The union of types and variables such that they can both be generalized and used in constraints.

**package** SIMF Conceptual Model::Rules

#### Associations

has multiplicity : [Multiplicity Constraint](#_4ff1432ad36ac8beb6cbb7e9323d9f24) [\*] *Subsets*: constrained by:[Rule](#_82919e40af9ad2e13647e9d37bbf0956)



Multiplicity constraint of a type or property in a context.

### Association Unique Set

**package** SIMF Conceptual Model::Rules

#### Association Ends

has unique : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1..\*] *Subsets*: constrained by:[Rule](#_82919e40af9ad2e13647e9d37bbf0956)



The set of involved roles within a situation/relation that uniquely identify an individual.

uniqueness constraint : [Uniqueness Constraint](#_982e84b7afc784b4d0aa763204953a3d) [\*] *Subsets*: constrained by:[Rule](#_82919e40af9ad2e13647e9d37bbf0956)



### Class Uniqueness Constraint

A constraint that, within the situation type the rule applies to, the set of instances bound to the set of types in the "has unique" relation must be unique and serves to define the "individuality" of each individual. A Uniqueness constraint that is not constrained by a situation indicates that the role may have only one member.  
Note: Uniqueness may be used to define a "key".

#### Direct Supertypes

[Type Constraint](#_ded47679f07683882f8f128d6911711a)

**package** SIMF Conceptual Model::Rules

#### Attributes

is primary identity : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)



A uniqueness constraint that can be interpreted as a "primary key"

#### Associations

has unique : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1..\*]

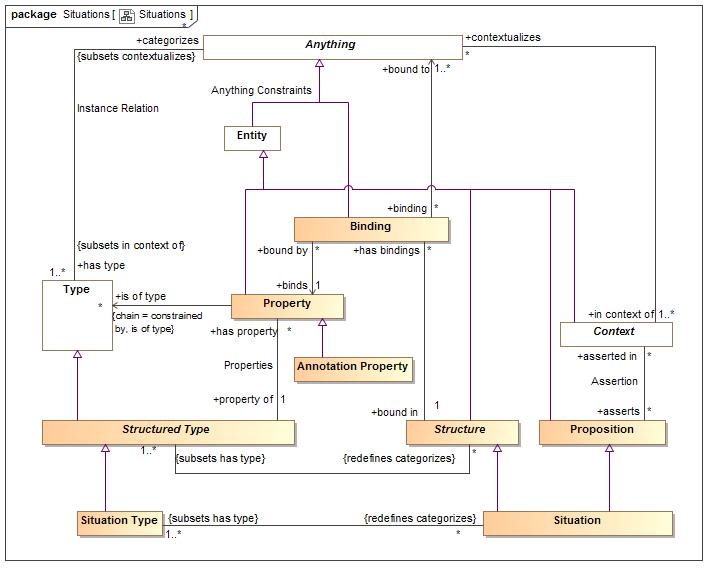


The set of involved roles within a situation/relation that uniquely identify an individual.

## SIMF Conceptual Model::Situations

Situations are arrangements of individuals, assertions and the relations and assertions between them over a timespan. Any "condition" that exists is a situation - including relations.

### Diagram: Situations



1. Situations

A set of role bindings about a specific subject

### Class Binding

A binding binds a particular individual to a property in a situation. A.K.A. the value of a property.  
  
For relations (which are situation), this is also known as an "association end".  
  
Where a binding is bound to a property, the binding shall bind to the bindings of that property.  
  
Where a binding is to an expression, the binding shall evaluate to the evaluation of the expression.  
  
The bound to individual must conform with the "as type" type of the property. If the bound individual conforms to the "requires type" of the property, the "has type" in the individual will be asserted.  
  
THe type of the "bound in" situation must (directly or indirectly) have the type the "binds" property is a property of.

#### Direct Supertypes

[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

**package** SIMF Conceptual Model::Situations

#### Associations

binds : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1]



The property a binding binds an individual to.

bound to : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [1..\*]



The individual(s) bound to a property in a specific situation.

bound in : [Structure](#_8c517cf1950741c0f89edebf828214cc) [1]



The situation owning the binding.

### Association bound in situation

**package** SIMF Conceptual Model::Situations

#### Association Ends

has bindings : [Binding](#_e829344c78ea1a9e5e18c7bc51ff8f64) [\*]



Bindings asserted in a situation.

bound in : [Structure](#_8c517cf1950741c0f89edebf828214cc) [1]



The situation owning the binding.

### Association bound individual

**package** SIMF Conceptual Model::Situations

#### Association Ends

bound to : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [1..\*]



The individual(s) bound to a property in a specific situation.

binding : [Binding](#_e829344c78ea1a9e5e18c7bc51ff8f64) [\*]



### Association bound property

**package** SIMF Conceptual Model::Situations

#### Association Ends

bound by : [Binding](#_e829344c78ea1a9e5e18c7bc51ff8f64) [\*]



binds : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1]



The property a binding binds an individual to.

### Association Properties

**package** SIMF Conceptual Model::Situations

#### Association Ends

has property : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [\*]



Variables of a situation type

property of : [Structured Type](#_50241f5936e61055293ca95f860768d8) [1]



Situation type defining a property.

### Class Property

The way in which instances of a type play a roles in a situation (including relationships). Sometimes called a variable, argument or role.  
  
In a conceptual model the <property> typically represents the "verb phrase" defining how instances of the involved type play a role an a situation.  
  
In a data structure the property is a "slot" of a record.  
  
In a function, it is the argument.

#### Direct Supertypes

[Entity](#_eb8398b5a178c638b98597120ec51c4d), [Type or Property](#_0ae2de066cc60f0d0ea185e32c2ee88c)

**package** SIMF Conceptual Model::Situations

#### Associations

property of : [Structured Type](#_50241f5936e61055293ca95f860768d8) [1]



Situation type defining a property.

: [Property Constraint](#_91be190e8014514a597300b286148d3e) [\*] *Subsets*: constrained by:[Rule](#_82919e40af9ad2e13647e9d37bbf0956)



has specialization : [Property Generalization Constraint](#_8453c07e3c7aaeab8e715f3569b782ab) [\*]



Generalizations of a property

has generalization : [Property Generalization Constraint](#_8453c07e3c7aaeab8e715f3569b782ab) [\*] *Subsets*: constrained by:[Rule](#_82919e40af9ad2e13647e9d37bbf0956)



Generalization/Specialization rules for a property.

is of type : [Type](#_dfe1514224ca21cedba7b2b29802db50) [\*]



A type of instances bound to a property. Also known as the "range" of a property.  
If asserted the property rule shall be owned and asserted by property.

### Class Situation

An identifiable arrangement of individuals, assertions and the relations and assertions between them over a timespan. Any condition or occurrence is a situation.  
  
The individuals involved via properties are bound to a situation via bindings.  
  
Syn. "State of affairs"

#### Direct Supertypes

[Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151), [Proposition](#_3bd7c7d249201ad6f2447c6d182ba7f1), [Structure](#_8c517cf1950741c0f89edebf828214cc)

**package** SIMF Conceptual Model::Situations

#### Associations

: [Situation Type](#_c568596a9f8653e157460ae519e96906) [1..\*] *Subsets*: has type:[Type](#_dfe1514224ca21cedba7b2b29802db50)



### Class Situation Type

A type of identifiable arrangement of individuals, assertions and the relations between them over a timespan. Elements of a situation may change over time.  
  
The role individuals play in a situation are identified as properties of the situation type.  
  
Syn. Type of a state of affairs.  
  
[BFO] Endurant

#### Direct Supertypes

[Structured Type](#_50241f5936e61055293ca95f860768d8)

**package** SIMF Conceptual Model::Situations

#### Associations

: [Situation](#_bed9725f6bf8a4fff6fd1087850b7259) [\*] *Redefines*: categorizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)



### Class Structure

A structure of elements where the structure is represented by a set of bindings that bind individuals to the structure with reference to a property.

#### Direct Supertypes

[Entity](#_eb8398b5a178c638b98597120ec51c4d)

**package** SIMF Conceptual Model::Situations

#### Associations

: [Structured Type](#_50241f5936e61055293ca95f860768d8) [1..\*] *Subsets*: has type:[Type](#_dfe1514224ca21cedba7b2b29802db50)



has bindings : [Binding](#_e829344c78ea1a9e5e18c7bc51ff8f64) [\*]



Bindings asserted in a situation.

### Class Structured Type

A type that has properties such that instances (structures), may bind individuals to structures based on properties.

#### Direct Supertypes

[Type](#_dfe1514224ca21cedba7b2b29802db50)

**package** SIMF Conceptual Model::Situations

#### Associations

: [Structure](#_8c517cf1950741c0f89edebf828214cc) [\*] *Redefines*: categorizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)



has property : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [\*]

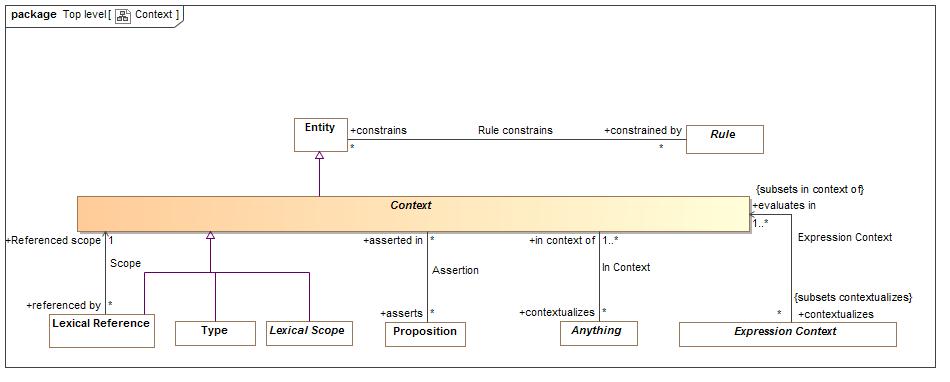


Variables of a situation type

## SIMF Conceptual Model::Top level

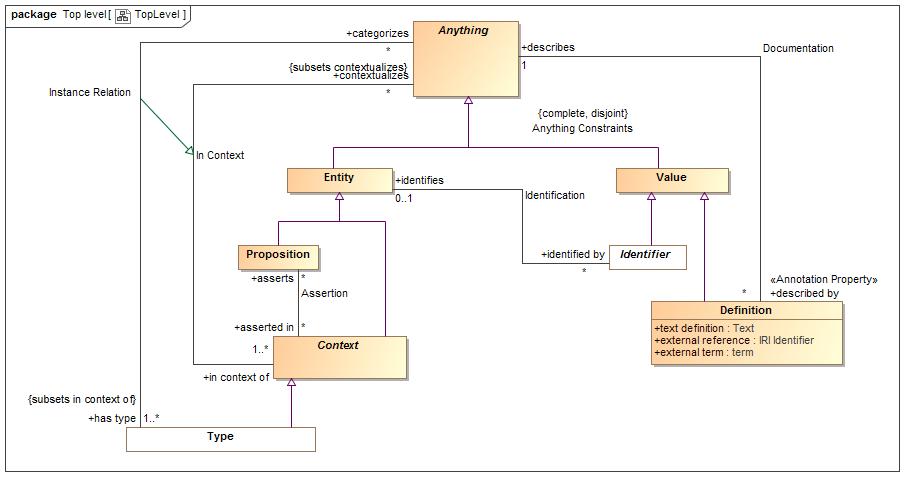
The top level objects provide the foundation for all objects in the model

### Diagram: Context



1. Context

### Diagram: TopLevel



1. TopLevel

### Class Anything

Any thing or value that does or may exist in any possible world. Anything is the supertype of all conceptual model types and may therefore participate in unbounded relations.

**package** SIMF Conceptual Model::Top level

#### Associations

described by : [Definition](#_1a6d88e097d757268d09f68af82fbd34) [\*]



Informal descriptions

has type : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1..\*] *Subsets*: in context of:[Context](#_66d62b068053cee3464e1e03e6035eed)



One of the types defining a concept

in context of : [Context](#_66d62b068053cee3464e1e03e6035eed) [1..\*]



The set of context that contextualizes a concept

defined in : [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151) [1] *Subsets*: in context of:[Context](#_66d62b068053cee3464e1e03e6035eed)



Lexical scope defining model elements.

### Association Assertion

**package** SIMF Conceptual Model::Top level

#### Association Ends

asserts : [Proposition](#_3bd7c7d249201ad6f2447c6d182ba7f1) [\*] *Subsets*: in context of:[Context](#_66d62b068053cee3464e1e03e6035eed)



All asserted situations (including bindings and relations) are true for anything contextualized by the context.

asserted in : [Context](#_66d62b068053cee3464e1e03e6035eed) [\*] *Subsets*: in context of:[Context](#_66d62b068053cee3464e1e03e6035eed)



Context in which the situation is considered true.

### Class Context

A grouping of concepts contextualized in some way and with some common attributes, rules and/or relations.  
  
A context provides a binding between a set of rules or assertions and the elements those rules or assertions apply to.

#### Direct Supertypes

[Entity](#_eb8398b5a178c638b98597120ec51c4d)

**package** SIMF Conceptual Model::Top level

#### Associations

contextualizes : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [\*]



The set of concepts "in" a context

asserts : [Proposition](#_3bd7c7d249201ad6f2447c6d182ba7f1) [\*]



All asserted situations (including bindings and relations) are true for anything contextualized by the context.

### Class Entity

A representation of the conception of anything other than values, this includes individuals, types, axioms, situations, speech acts, information structures, etc.  
[todo] identity

#### Direct Supertypes

[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

**package** SIMF Conceptual Model::Top level

#### Associations

identified by : [Identifier](#_18f8ef1b23e6cdf9278bd94f24f73c26) [\*]



Names for a concept

has preferred term : [Term](#_1945edd0888993a52c5dc6467a7b3ef8) [0..1] *Subsets*: identified by:[Identifier](#_18f8ef1b23e6cdf9278bd94f24f73c26)



Default term to use for an entity

constrained by : [Rule](#_82919e40af9ad2e13647e9d37bbf0956) [\*]



Rules applying to things contextualized by a context.

### Association In Context

The association between a context and the set of concepts contextualized by that context.

**package** SIMF Conceptual Model::Top level

#### Association Ends

contextualizes : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [\*]



The set of concepts "in" a context

in context of : [Context](#_66d62b068053cee3464e1e03e6035eed) [1..\*]



The set of context that contextualizes a concept

### Class Proposition

A statement or condition with a truth value that can be determined or asserted.

#### Direct Supertypes

[Entity](#_eb8398b5a178c638b98597120ec51c4d)

**package** SIMF Conceptual Model::Top level

#### Associations

asserted in : [Context](#_66d62b068053cee3464e1e03e6035eed) [\*]



Context in which the situation is considered true.

### Association Term Preference

**package** SIMF Conceptual Model::Top level

#### Association Ends

has preferred term : [Term](#_1945edd0888993a52c5dc6467a7b3ef8) [0..1]



Default term to use for an entity

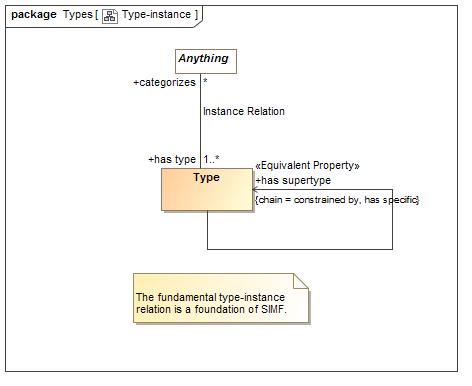
preferred for : [Entity](#_eb8398b5a178c638b98597120ec51c4d) [0..1]



## SIMF Conceptual Model::Types

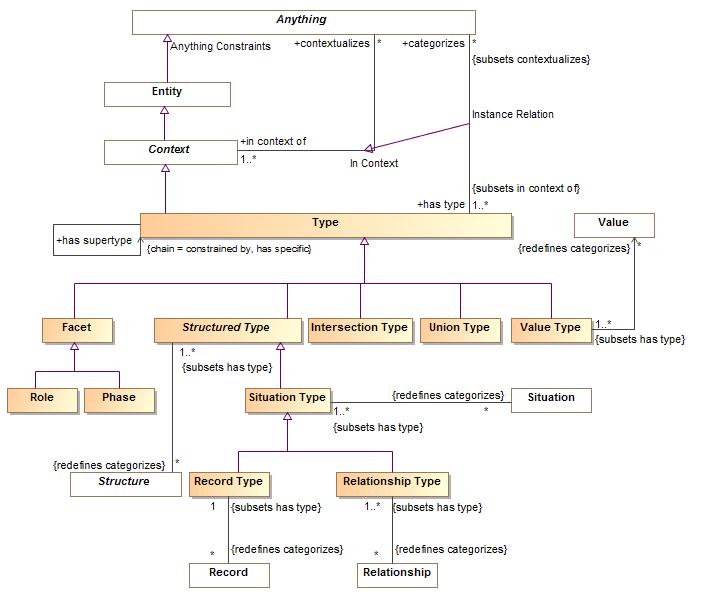
Type provide for ways to categorize anything based on what it is, the roles it plays or the phases it may be in.  
Something may be categorized by any number of types (multiple classification assumption).

### Diagram: Type-instance



1. Type-instance

### Diagram: Types



1. Types

### Class Facet

A facet is a "mix in" type that defines some aspect of something but does not represent the "fundamental type" of that thing, but some potentially transient role, phase or other way to classify. Something must have at least one type that is not a facet.  
Facets do not define independent identity.  
In instance of a facet must also have a type that is not a facet to provide the identity of the instance.  
  
The type(s) a facet may categorize may be constrained by setting the "as facet" property of Generalization. E.g. Policeman is a role of a person.  
  
BFO: Non-rigid sortal

#### Direct Supertypes

[Type](#_dfe1514224ca21cedba7b2b29802db50)

**package** SIMF Conceptual Model::Types

### Association Instance Relation

The relation between a type and the concepts that type categorizes, the instances

### Direct Supertypes

[In Context](#_52c887644007b8e51a1f6e976113707a)

**package** SIMF Conceptual Model::Types

#### Association Ends

categorizes : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [\*]



The set of concepts of individuals described by a type.

has type : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1..\*]



One of the types defining a concept

### Class Intersection Type

An intersection is a type that has an extent which is the complete intersection of the extents of all supertypes. Intersection is a stronger statement than a subtype as a subtype may not be a complete intersection. If an instance has all supertypes the intersection type will be inferred.

#### Direct Supertypes

[Type](#_dfe1514224ca21cedba7b2b29802db50)

**package** SIMF Conceptual Model::Types

### Class Phase

A phase (or state) is a characteristic of something that exists for limited time(s). Something takes on or looses a phase as a result of some event. E,g, Teenager, living, closed invoice.

#### Direct Supertypes

[Facet](#_3b2e69eb6121d1e3a1180bbe8ee64013)

**package** SIMF Conceptual Model::Types

### Class Role

A role is a type facet type that defines a specific purpose or behavior of a class of things. E.g. teacher, policeman, employer.

#### Direct Supertypes

[Facet](#_3b2e69eb6121d1e3a1180bbe8ee64013)

**package** SIMF Conceptual Model::Types

### Class Type

A categorization of concepts about individuals or types based on specific criteria

#### Direct Supertypes

[Context](#_66d62b068053cee3464e1e03e6035eed), [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151), [Type or Property](#_0ae2de066cc60f0d0ea185e32c2ee88c)

**package** SIMF Conceptual Model::Types

#### Associations

categorizes : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [\*] *Subsets*: contextualizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)



The set of concepts of individuals described by a type.

has generalization : [Type Generalization Constraint](#_5f3998cf1a072f724861db93cee66cbf) [\*] *Subsets*: constrained by:[Rule](#_82919e40af9ad2e13647e9d37bbf0956)



Generalizations of a type

has specialization : [Type Generalization Constraint](#_5f3998cf1a072f724861db93cee66cbf) [\*]



Generalization/Specialization rules for a type.

types : [Property Type Constraint](#_3e4d7d36a3b2cb4c9b85c1b88930178a) [\*]



Properties typed by a type

has supertype : [Property Type Constraint](#_3e4d7d36a3b2cb4c9b85c1b88930178a) [\*]



Supertypes(s) of a type as defined by generalization rules.  
  
If asserted the generalization rule shall be owned and asserted by subtype.

has covering : [Covering Constraint](#_507049575ebfa9f535e8f25db14a0760) [\*]



Covering constraints of a type.

### Class Union Type

A Union is any type that has an extent which is the complete union of the extents of all context that specialize the Union.   
  
Note: The semantic different between a union and a supertype is not clear but it is included for coverage of FOL.

#### Direct Supertypes

[Type](#_dfe1514224ca21cedba7b2b29802db50)

**package** SIMF Conceptual Model::Types

## SIMF Conceptual Model::Values

The values package defines the concepts of values and quantities expressed in units.

Values may be differentiated from entities in that values have no independent lifetime or "identity" other than the value its self. E.g. the number 5 "just is" and can't be changed. Properties and relations referencing values can, of course, change but the values are constant.

The failure to properly express units in data models often results in errors, inefficiencies and risk. Translation and federations between models, schema and data sources that is not cognizant of the units used would be even more error prone and risky. For example, what does “Speed limit 50” mean? For these reasons the SIMF language provides specific support for specifying quantity kinds and unit types in conceptual, logical and physical models. The SIMF mapping rules may then perform the appropriate unit conversions.

The foundation of information specification in SIMF at all levels is the type system. Types specified for all properties and relations involving values must match the types of the related values. The concepts of units and values as defined in "VIM" [JCGM 200-2008] is used as the basis for defining the types used in SIMF to guarantee type safety of quantities across different representations. Since many existing models and schema do not include well defined units some effort may be required to find and then specify the implicit units based on documentation, SME interviews or inspection of data or source code. It is recommended that the units used by external models and schema be determined prior to attempting federation and integration of information based on those models or schema.

**VIM [JCGM 200-2008] concepts of quantities and units**

quantity: property of a phenomenon, body, or substance, where the property has a magnitude that can be expressed as a number and a reference [ed. to a unit]

kind of quantity (kind): aspect common to mutually comparable quantities

measurement unit (unit): real scalar quantity, defined and adopted by convention, with which any other quantity of the same kind can be compared to express the ratio of the two quantities as a number

**SIMF concepts of quantities and units**

SIMF uses the VIM concepts to define "quantity values" and types to capture the quantity kind and unit. The goals for this type based approach are:

* That it is clearly grounded in semantics as defined in VIM
* That a type may be used to specify the range of a property or relation involving unit based values.
* That a quantity value (e.g. 5 grams) be representable as a simple number with a type.
* That there is a clear type hierarchy starting with a representationally independent type in a conceptual model (e.g. mass) that can be further specialized to a specific unit in a logical model (e.g. grams) and further specialized to be represented by a physical data type (e.g. “double”).
* That external models and schema may have unit specifications asserted without changing the schema.
* That a quantity of an entity be able to be referenced without a specific quantity value being known (e.g. John’s weight).
* That systems of units such as [ISO-80000] or [OMG QUDV] (A part of SysML) be able to be directly referenced as the definition of a unit.

SIMF defines three types to realize the above goals: Quantity Kind, Unit Type, Base Unit Type. SIMF also defines Quantity Values, which are instances of unit types.

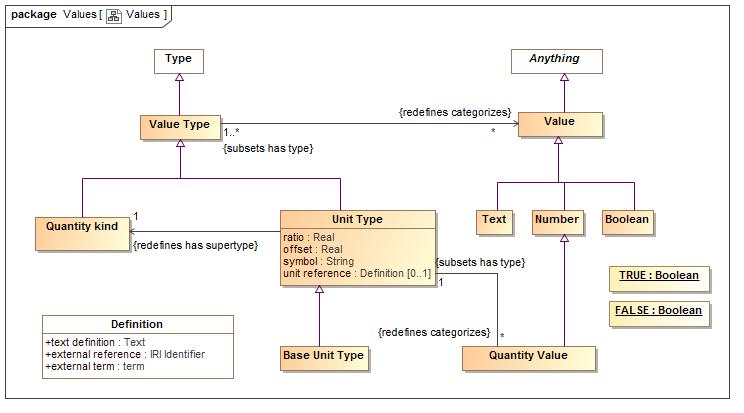
In VIM a quantity has a magnitude that is expressed as a number and a reference. The SIMF quantity value is the numeric value of such a quantity where the reference is specified by the “unit reference” property of the quantity value’s type. The quantity value’s type is a “Unit Type”. The Unit type has attributes for converting a unit to a base unit, a symbol and a unit reference. Based on VIM the unit reference may be “a measurement unit, a measurement procedure, a reference material, or a combination of such” and is specified with a description that contains reference information. In summary, the reference of a SIMF quantity value is determined indirectly through its unit type. A quantity value has exactly one unit type and exactly one Quantity Kind. A quantity value expressed in any unit of the same quantity kind may be converted to any other unit of the same quantity kind.

This type-based sapproach allows specification of a property at the conceptual (quantity kind) logical (unit type) or physical (unit type with a numeric type) levels. Such specifications use the same type-based approach used for other aspects of the models. Given this information a SIMF implementation may correctly and reliably convert between compatible types regardless of representation. Please see the specification of the value types, attributes and relationships for more detail.

**Example:**

* A specification for a road segment has a property “Speed limit”.
* The type of this property in a reference conceptual model is “Speed:Quantity Kind”.
* A unit “Kilometer per Hour:Unit Type” is defined as a subtype of “Speed:Quantity Kind” with a “unit reference” of “[ISO-80000.4] Kilometer per Hour”. Note that quantity kinds and unit types would normally be defined in reference models that correspond to a “system of units”.
* Miles per hour is also defined as a subtype of Speed.
* A physical schema defines “Speed-KPH: Integer”.
* A SIMF mapping rule maps “Speed limit” to “Speed-KPH” and asserts a type of “Kilometer per Hour” on the “Speed-KPH” end.
* A data file defines a road “Route One” with a speed limit of 100:KPH-Int.
* When converted to a U.S. application this speed limit of route one can be viewed as 62:MPH-Int.

### Diagram: Values



1. Values

### Class Base Unit Type

One unit type of a quantity kind may be marked as the base unit. The base unit provides the basis for conversions between units of the same quantity kind. The base unit always has a ratio of one and an offset of zero.

#### Direct Supertypes

[Unit Type](#_9a97d5f73bf658c81147f5fab194bf88)

**package** SIMF Conceptual Model::Values

### Class Boolean

A value that may be true or false.

#### Direct Supertypes

[Value](#_a739673c8d53da123e392b7e5059ceec)

**package** SIMF Conceptual Model::Values

### Class Number

The concept of a number that may be expressed in any way.

#### Direct Supertypes

[Value](#_a739673c8d53da123e392b7e5059ceec)

**package** SIMF Conceptual Model::Values

### Class Quantity kind

[JCGM 200:2008] A Quantity Kind is an aspect common to mutually comparable quantities represented by one or more units. Units with a common quantity kind may be algorithmically converted to any other unit of that quantity kind. e.g. temperature.  
  
Quantity kinds are a supertype of unit types which are then a type of all quantity values, Quantity values are mutually comparable with all other quantity values categorized by the same quantity kind.

#### Direct Supertypes

[Value Type](#_b08132d9b30f1d47632a28aa6e4894bf)

**package** SIMF Conceptual Model::Values

### Class Quantity Value

A quantity value is a numeric magnitude with a unit type that may be used as the value of a quantity property as defined by [JCGM 200:2008]. The reference of the quantity is defined by the "unit reference" property of the Unit Type.  
  
Each quantity value has exactly one Quantity Kind as a type.  
  
In a physical model a quantity value must have a type that specifies its unit (e.g. "Gram") and may have a data type specifying its numeric representation (e.g. "Double").  
  
[JCGM 200:2008] A quantity is a property of a phenomenon, body, or substance, where the property has a magnitude that can be expressed as a number and a reference.   
  
Note: A quantity as defined here is a scalar. However, a vector or a tensor, the components of which are quantities, is also considered to be a quantity.  
  
e.g. 5cm is an instance of the unity type "Centimeter"

#### Direct Supertypes

[Number](#_f5a86db7bd9636f0fa472c3859bc9c3c)

**package** SIMF Conceptual Model::Values

#### Associations

: [Unit Type](#_9a97d5f73bf658c81147f5fab194bf88) [1] *Subsets*: has type:[Type](#_dfe1514224ca21cedba7b2b29802db50)



### Class Text

Text is a value represented using symbols which have a meaning to stakeholders but otherwise have no formal semantic implication. Properties involving values may have a semantic implication.

#### Direct Supertypes

[Value](#_a739673c8d53da123e392b7e5059ceec)

**package** SIMF Conceptual Model::Values

### Class Unit Type

A Unit type is a type of a quantity value referencing a specific unit and a required type of a property representing a quantity.   
  
Each quantity value has a reference as defined by the "unit reference" property of the quantity value's type.  
  
[JCGM 200:2008] A Unit is a real scalar quantity, defined and adopted by convention, with which any other quantity of the same quantity kind can be compared to express the ratio of the two quantities as a number. e.g. Degrees Centigrade, Miles.  
  
Each unit type represents refinement of a quantity kind using generalization and is thus substitutable for that quantity kind. Typically quantity kinds are used in conceptual models and unit types in physical or logical models.  
  
Unit types may only subtype quantity kinds.  
  
Note that unit types are not units, but the type of quantity values expressed in a common unit as defined in [JCGM 200:2008].

#### Direct Supertypes

[Value Type](#_b08132d9b30f1d47632a28aa6e4894bf)

**package** SIMF Conceptual Model::Values

#### Attributes

ratio : [Real](#_aef4bcae5ebc35dd9653214547b3e3cc)



the multiplier by which to multiple the referenced unit to convert to the base unit

offset : [Real](#_aef4bcae5ebc35dd9653214547b3e3cc)



the difference between zero in the referenced unit and zero in the base unit after the ratio is applied

symbol : [String](#_e8a6ce315d976318da3ab784a645ea44)



the accepted symbol for the unit referenced by the unit type

unit reference : [Definition](#_1a6d88e097d757268d09f68af82fbd34) [0..1]



The unit reference is the reference to a unit shared by all quantities that are instances of a unit type.  
  
[JCGM 200:2008] A reference can be a measurement unit, a measurement procedure, a reference material, or a combination of such. For magnitude of a quantity.  
  
Typical references include ISO 8000 and OMG QUDV.

#### Associations

: [Quantity Value](#_746074a5bfbd6e26906da5d4bd0d2a7f) [\*] *Redefines*: categorizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)



: [Quantity kind](#_ca1c56b440439615024c837658185d15) [1] *Redefines*: has supertype:[Type](#_dfe1514224ca21cedba7b2b29802db50)



### Class Value

A value is an atomic piece of information without a specific lifetime or identity independent of the value. Values include numbers, strings and other atomic "primitive" data.  
  
In UML values may be defined by the name of an instance specification with a value type.

#### Direct Supertypes

[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

**package** SIMF Conceptual Model::Values

### Class Value Type

A type categorizing values where a value is an atomic piece of information without a specific lifetime or identity independent of that value. Values include numbers, strings and other atomic "primitive" data.

#### Direct Supertypes

[Type](#_dfe1514224ca21cedba7b2b29802db50)

**package** SIMF Conceptual Model::Values

#### Associations

: [Value](#_a739673c8d53da123e392b7e5059ceec) [\*] *Redefines*: categorizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)



# Foundational Assumptions (Normative)

## Multiple representations of overlapping concepts

A base assumption of SIMF is that there are and will be multiple information resources that represent information about the same and overlapping things. These different representations may use different definitional languages (e.g. XML Schema, SQL, RDF, UML) and well as different domain terminology and human languages (e.g. Risk, Danger, Achtung, Risque). Our task is to federate information from multiple sources as well as express information from one source in another form.

## Models include data

Any structured set of information about anything is considered a “model”. This usage of the term model is consistent with the ontological usage but is an expansion of the term as typically used in modeling languages such as UML.

## Conceptual Models

For a SIMF implementation there will exist a conceptual model that contains a all the concepts to be interpreted and various mappings to physical or logical data representations. We will use the label CONCEPT and REPRESENTATION thorough to represent elements at both levels. It should also be noted that REPRESENTATION may or may not be fully physical, some REPRESENTATIONs may, in turn, have various syntaxes. For example, there are multiple syntaxes for RDF. We are treating all these layers of REPRESENTATION the same with the assumption there is an isomorphic transform between elements in the same REPRESENTATION family (e.g. the RDF logical model and RDF in Turtle or JSON-LD Syntax). The conceptual model will be the “hub” or “pivot point” to integrate multiple REPRESENTATION models.

Conceptual models are intended to be models of the world – real world things. Facts about real world things are represented by data models in a data REPRESENTATION form but these facts also have a CONCEPT counterpart. While any implementation of a conceptual model must have some representation and/or behavior, the representation is abstract in terms of the specification (there is no normative syntax for data about instances of a conceptual model).

There are many possible implementation choices for CONCEPT models. Implementations MAY use the conceptual model only as a “virtual” rule set and have no physical representation of information about CONCEPT instances. Implementations MAY instantiate a physical representation corresponding directly to the conceptual model.

There is no assumption of any single or distinguished conceptual model. CONCEPT MODEL is a role for a model, some conceptual models may be derived from existing REPRESENTATIONs – particularly model based representations such as UML or OWL. Of course, some models will make better conceptual hubs than others. The Threat/RISK model is designed to be a conceptual model and will be refactored as required to represent the set of concepts represented by the REPRESENTATIONs it is intended to federate

## Identity and identifiers

There are different forms of identity and identifiers. There is the identity of the “real world” thing that some representation is stating facts about and there is the identity of the representation (the data or assertion - facts or statements). An identifier for a thing is called its “sign”. The sign of the real-world thing and the sign of data about that thing are not the same. Most REPRESENTATION models have representation identifiers but sometimes confuse these as signs for (identifiers for) the real world thing. Further, anything may have multiple identifiers – however many technologies only have one. “Keys” tend to be signs for real things, but this is not consistent.

## Facts

The term “fact” as used here is based on “fact modeling” but does not necessarily mean that something is “true”, a fact is some statement that asserts something - it may or may not be true and that truth may be contextual. Other terms could be “assertion” or “statement”.

## Representations of a CONCEPT

Where there exists some CONCEPT of something there is a REPRESENTATION data element representing facts about this concept. The concept is also a representation is some respects, but this representation is playing the role of representing the real-world thing, not a data structure.

### Relation

* Verb form: representation pattern🡪represents🡪CONCEPT pattern
* Noun form: Representation(representation pattern, CONCEPT pattern)

“Represents” provides the basis for the relation between a REPRESENTATION and a CONCEPT. Both ends or the represents relation require an identity for the “fact” or “facts” being represented. Thus “represents” depends on the ability to identify both the REPRESENTATION and CONCEPT fact. See 8.14.7.

The represents relation asserts that the REPRESENTATION element represents information about the CONCEPT. The information is not assumed to be complete or consistent. For various types of concepts this implies:

* **Individuals:** That the REPRESENTATION individual is a sign for facts (data) about the CONCEPT individual. Individuals are identifiable instances where the extent of the sign represents a single “thing” in the world. Types, context, relations and values are not individuals.
* **Types**: That the REPRESENTATION type describes the CONCEPT type. The extent of the REPRESENTATION type MAY be a subset of the extent of the CONCEPT type.
* **Context**: That the REPRESENTATION context contextualizes all or a subset of the CONCEPT context.
* **Relations**: That the relation semantics holds true for the bound elements of both the REPRESENTATION and CONCEPT

## Representation identifiers

A REPRESENTATION resource contains a set of facts, such a resource may change over time so the set of facts may also change. At a particular time, as seen from a “transaction[[3]](#footnote-3)”, a resource has a fixed set of facts. Within a transaction a particular fact has an IDENTITY. The IDENTITY may be the same across multiple transactions from the same resource. The combination of the identity of the transaction and the identity of the fact within that transaction must be universally unique.

A fact in a REPRESENTATION resource has an IDENTIFIER. The form and structure of that identifier is technology dependent and not determined in this specification but it must be representable as a text string. Given a transaction/fact identifier combination any fact must be traceable to its origin. Where this pattern is replicated to the source of the fact, full provenance of the information is provided. A REPRESENTATION model may provide additional provenance information as part of a transaction and/or resource source reference but the semantics of that provenance is not assured unless the source is also a SIMF model.

Identical REPRESENTATION identifiers, within the same resource, are assumed to identify the same concept. Identical identifiers within different resources MAY be interpreted as identifying the same concept. Such identifiers are frequently called UUIDs or GUIDS.

### Example physical identifiers

The STIX fragment:

<stix:TTP id="opensource:ttp-62ff7599-e1db-414f-b742-908def2ec219"

timestamp="2014-11-06T17:46:28.913306+00:00"

xsi:type="ttp:TTPType" version="1.1.1">

<ttp:Title>ZeuS</ttp:Title>

</stix:TTP>

Could use an XPATH expression for the ID, such as: ‘//\*[@ID=”opensource:ttp-62ff7599-e1db-414f-b742-908def2ec219]’ as the physical ID for the TTP fact (asserting the existence of the TTP) and "‘//\*[@ID=”opensource:ttp-62ff7599-e1db-414f-b742-908def2ec219”:]/Title[1]’ as the ID for the title fact.

Physical identifiers are not typically exposed to the user.

## CONCEPT representation identifiers

All conceptual facts, or “CONCEPTS” reside in some knowledge base, this knowledge base has a unique physical and logical identity. Within a knowledge base there exists a set of concepts with an identifier unique within that knowledge base. The combination of the knowledge base identifier and the concept identifier must be universally unique.

## Sources

All concepts come from some source outside of the conceptual system (even if this is a rule or inference operating in the conceptual system). Concepts have a SOURCE. All sources produce concept representations via some rule (inferences are considered rules). A REPRESENTATION of a concept (represents relation) is one such rule. Thus all concepts may be traced to their source as it existed at a particular time and was interpreted or computed based on a rule at that time.

### Relation:

* Verb form: Fact🡪has source🡪Source Record
* Noun form: Source(Fact, Source Record)

Every conceptual fact has a source, that source identifies:

* The transaction. Transactions identify any physical resource as well as the time of the transaction.
* The rule that produced the fact
* The REPRESENTATION source identifier(s)

Since facts come from an identifiable source, that source and any internal structure MAY be used as a hint in producing the fact identifier. For resources with internal structure, this would typically be the Resource followed by a set of Packages followed by some name, ID or path of the REPRESENTATION fact.

The same source MAY result in multiple concepts. A representation concept is unique to a source but may be declared as equivalent to other representations concept.

## Ownership

Representation information shall be “owned” directly or indirectly by a model, this ownership shall correspond with the structure of the source where such structure is important in interpreting the concept. The statement is owned, the ownership of the statement has no semantic interpretation with relation to what the concept means unless there are statements about the owning context. Ownership of the concept may have a semantic interpretation as to the authority, scope or timeframe of the concept.

Ownership is one dimension of “context”, a concept may be contextualized by any number of context. Context may impact the interpretation of or trust in a concept. The context set of an owner in included in the context sets of concepts it owns.

## Lifetime and context of facts

Logically concepts and statements are never “deleted”, they may become irrelevant or dated as would be determined by statements about the concept. Implementations MAY optimize a knowledge base by deleting facts that have been determined to be irrelevant for all purposes of the knowledge base (i.e. they would never impact the truth of any computation or query). Deletion of facts is not, in general, recommended (deletion is so 20th century).

Example: the fact “Thomas Woodrow Wilson is president of the United States” was true “March 4, 1913 through March 4, 1921”. We accept it will always be true that this fact was true in this timeframe, so there is no sense in “deleting” the fact. The timeframe fact “March 4, 1913 through March 4, 1921” is a fact about the fact: “Thomas Woodrow Wilson is president of the United States”. Another fact about this fact is that its source resource was <http://www.whitehouse.gov> on January 14th, 2015 at 9:05PM-EST. This “source” fact is classified as an “Annotation” which represents metadata.

Note that the above implies that there may be more than one FACT that says exactly the same thing, but they may come from different sources or be stated in different context or timeframes. Or, different sources or context may assert conflicting facts about the same thing. Trust is a function of the interpretation of a set of facts, including their source, timeframe and context. Interpretation is time dependent (i.e. transactional), with respect to a given set of context operating on a given knowledge base at a given time. (Note that a knowledge base may include all or part of the contents of other repositories and that repositories may be physical or virtual). A knowledge base could be as small as one or more files on your tablet or as large as Google or Wikipedia.

The UML profile is the only defined graphical notation.

# Annex A Examples (informative)

**The SIMF submission calls for worked examples. The worked examples for the SIMF initial submission is the initial submission for an operational threat and risk model, document: sysa/2015-11-01**

# Mapping to OWL 2 (normative)

Examples are given below that show the transformation of UML modeled in SIMF to an exported OWL 2 ontology. The OWL ontologies are presented in OWL Functional Syntax.

The first diagram below, for a simple UML class, shows the ontology is transformed as the package containing the UML class. Subsequent diagrams do not show the package in the diagram for the sake of brevity.

## Class



Ontology(<http://nomagic.com/ontology/example-case/case-01>

Declaration(

Class(:Person)

)

AnnotationAssertion(rdfs:label :Person "Person"@en)

)

## Class Generalization



Ontology(<http://nomagic.com/ontology/example-case/case-04>

Declaration(

Class(:FutsalTeam)

)

Declaration(

Class(:SoccerTeam)

)

AnnotationAssertion(rdfs:label :FutsalTeam "Futsal Team"@en)

SubClassOf(:FutsalTeam :SoccerTeam)

AnnotationAssertion(rdfs:label :SoccerTeam "Soccer Team"@en)

)

## Class with Datatype Property



Ontology(<http://nomagic.com/ontology/example-case/case-02>

Import(<http://www.omg.org/spec/PrimitiveTypes/20100901>)

Declaration(

Class(:Person)

)

Declaration(

DataProperty(:hasName)

)

Declaration(

AnnotationProperty(<http://purl.org/dc/terms/description>)

)

Declaration(

Datatype(xsd:string)

)

AnnotationAssertion(rdfs:label :Person "Person"@en)

SubClassOf(

:Person

ObjectIntersectionOf(

DataMaxCardinality(1 :hasName xsd:string)

DataMinCardinality(1 :hasName xsd:string)

)

)

AnnotationAssertion(rdfs:label :hasName "has name"@en)

DataPropertyDomain(:hasName :Person)

DataPropertyRange(:hasName xsd:string)

AnnotationAssertion(<http://purl.org/dc/terms/description> <http://www.omg.org/spec/PrimitiveTypes/20100901#String> "An instance of String defines a piece of text. The semantics of the string itself depends on its purpose, it can be a comment, computational language expression, OCL expression, etc. It is used for String attributes and String expressions in the metamodel."@en)

)

## Class with Self-Referential Object Property



Ontology(<http://nomagic.com/ontology/example-case/case-02a>

Declaration(

Class(:Person)

)

Declaration(

ObjectProperty(:isRelatedTo)

)

AnnotationAssertion(rdfs:label :Person "Person"@en)

SubClassOf(

:Person

ObjectIntersectionOf(

ObjectMinCardinality(1 :isRelatedTo :Person)

)

)

AnnotationAssertion(rdfs:label :isRelatedTo "is related to"@en)

ObjectPropertyDomain(:isRelatedTo :Person)

ObjectPropertyRange(:isRelatedTo :Person)

)

## Class with Object Property



Ontology(<http://nomagic.com/ontology/example-case/case-03>

Declaration(

Class(:SoccerPlayer)

)

Declaration(

Class(:SoccerTeam)

)

Declaration(

ObjectProperty(:consistsOf)

)

AnnotationAssertion(rdfs:label :SoccerPlayer "Soccer Player"@en)

AnnotationAssertion(rdfs:label :SoccerTeam "Soccer Team"@en)

SubClassOf(

:SoccerTeam

ObjectIntersectionOf(

ObjectMaxCardinality(11 :consistsOf :SoccerPlayer) ObjectMinCardinality(5 :consistsOf :SoccerPlayer)

)

)

AnnotationAssertion(rdfs:label :consistsOf "consists of"@en)

ObjectPropertyDomain(:consistsOf :SoccerTeam)

ObjectPropertyRange(:consistsOf :SoccerPlayer)

)

## Property Holder with Datatype Property



Ontology(<http://nomagic.com/ontology/example-case/case-03a>

Import(<http://www.omg.org/spec/PrimitiveTypes/20100901>)

Declaration(

DataProperty(:hasName)

)

Declaration(

AnnotationProperty(<http://purl.org/dc/terms/description>)

)

Declaration(

Datatype(xsd:string)

)

SubClassOf(

owl:Thing

ObjectIntersectionOf(

DataMaxCardinality(3 :hasName xsd:string)

DataMinCardinality(2 :hasName xsd:string)

)

)

AnnotationAssertion(rdfs:label :hasName "has name"@en)

DataPropertyRange(:hasName xsd:string)

AnnotationAssertion(<http://purl.org/dc/terms/description>

<http://www.omg.org/spec/PrimitiveTypes/20100901#String> "An instance of String defines a piece of text. The semantics of the string itself depends on its purpose, it can be a comment, computational language expression, OCL expression, etc. It is used for String attributes and String expressions in the metamodel."@en)

)

## Property Holder with Self-Referential Object Property



Ontology(<http://nomagic.com/ontology/example-case/case-03b>

Declaration(

ObjectProperty(:isRelatedTo)

)

SubClassOf(

owl:Thing

ObjectIntersectionOf(

ObjectMinCardinality(1 :isRelatedTo)

)

)

AnnotationAssertion(rdfs:label :isRelatedTo "is related to"@en)

)

## Property Holder with Object Property



Ontology(<http://nomagic.com/ontology/example-case/case-03c>

Declaration(

Class(:Liquid)

)

Declaration(

ObjectProperty(:isDissolvedBy)

)

AnnotationAssertion(rdfs:label :Liquid "Liquid"@en)

SubClassOf(

owl:Thing

ObjectIntersectionOf(

ObjectMinCardinality(1 :isDissolvedBy :Liquid)

)

)

AnnotationAssertion(rdfs:label :isDissolvedBy "is dissolved by"@en)

ObjectPropertyRange(:isDissolvedBy :Liquid)

)

## Class with Object Property without Range



Ontology(<http://nomagic.com/ontology/example-case/case-03d>

Declaration(

Class(:Receptacle)

)

Declaration(

ObjectProperty(:holds)

)

AnnotationAssertion(rdfs:label :Receptacle "Receptacle"@en)

AnnotationAssertion(rdfs:label :holds "holds"@en)

ObjectPropertyDomain(:holds :Receptacle)

)

## Class with Subproperty



Ontology(<http://nomagic.com/ontology/example-case/case-05>

Declaration(

Class(:FutsalPlayer)

)

Declaration(

Class(:FutsalTeam)

)

Declaration(

Class(:SoccerPlayer)

)

Declaration(

Class(:SoccerTeam)

)

Declaration(

ObjectProperty(:composedOf)

)

Declaration(

ObjectProperty(:consistsOf)

)

AnnotationAssertion(rdfs:label :FutsalPlayer "Futsal Player"@en)

SubClassOf(:FutsalPlayer :SoccerPlayer)

AnnotationAssertion(rdfs:label :FutsalTeam "Futsal Team"@en)

SubClassOf(:FutsalTeam :SoccerTeam)

SubClassOf(

:FutsalTeam

ObjectIntersectionOf(

ObjectMaxCardinality(5 :composedOf :FutsalPlayer) ObjectMinCardinality(5 :composedOf :FutsalPlayer)

)

)

AnnotationAssertion(rdfs:label :SoccerPlayer "Soccer Player"@en)

AnnotationAssertion(rdfs:label :SoccerTeam "Soccer Team"@en)

SubClassOf(

:SoccerTeam

ObjectIntersectionOf(

ObjectMaxCardinality(11 :consistsOf :SoccerPlayer) ObjectMinCardinality(5 :consistsOf :SoccerPlayer)

)

)

AnnotationAssertion(rdfs:label :composedOf "composed of"@en)

SubObjectPropertyOf(:composedOf :consistsOf)

ObjectPropertyDomain(:composedOf :FutsalTeam)

ObjectPropertyRange(:composedOf :FutsalPlayer)

AnnotationAssertion(rdfs:label :consistsOf "consists of"@en)

ObjectPropertyDomain(:consistsOf :SoccerTeam)

ObjectPropertyRange(:consistsOf :SoccerPlayer)

)

## Class with Universal Quantification Constraint on Property I



Ontology(<http://nomagic.com/ontology/example-case/case-06>

Declaration(

Class(:Dog)

)

Declaration(

Class(:DogOwner)

)

Declaration(

Class(:Person)

)

Declaration(

Class(:Pet)

)

Declaration(

ObjectProperty(:has)

)

AnnotationAssertion(rdfs:label :Dog "Dog"@en)

SubClassOf(:Dog :Pet)

AnnotationAssertion(rdfs:label :DogOwner "Dog Owner"@en)

SubClassOf(:DogOwner :Person)

SubClassOf(

:DogOwner

ObjectIntersectionOf(

ObjectMinCardinality(1 :has :Dog)

ObjectAllValuesFrom(:has :Dog)

)

)

AnnotationAssertion(rdfs:label :Person "Person"@en)

AnnotationAssertion(rdfs:label :Pet "Pet"@en)

AnnotationAssertion(rdfs:label :has "has"@en)

ObjectPropertyDomain(:has :Person)

ObjectPropertyRange(:has :Pet)

)

## Class with Universal Quantification Constraint on Property II

This example differs from the previous example primarily in that the superclasses “Person” and “Pet” are from a different package than their subclasses “Dog Lover” and “Dog,” respectively. This is reflected in the OWL ontology by the import of this namespace.

The superclasses “Person” and “Pet”, defined in the package “Case 06”, are a different color and a lighter shade than the classes defined in the package “Case 07”. This is to distinguish them from the classes defined in this package. MagicDraw’s AutoStyler plugin can automatically set the properties for classes and other UML elements “defined elsewhere,” that is in a package not containing the defining diagram for the UML element (See section 2.2, Automatic Styling of Concept Models.).



Ontology(<http://nomagic.com/ontology/example-case/case-07>

Import(<http://nomagic.com/ontology/example-case/case-06>)

Declaration(

Class(<http://nomagic.com/ontology/example-case/case-06#Person>)

)

Declaration(

Class(<http://nomagic.com/ontology/example-case/case-06#Pet>)

)

Declaration(

Class(:Dog)

)

Declaration(

Class(:DogLover)

)

Declaration(

ObjectProperty(<http://nomagic.com/ontology/example-case/case-06#has>)

)

AnnotationAssertion(rdfs:label :Dog "Dog"@en)

SubClassOf(:Dog <http://nomagic.com/ontology/example-case/case-06#Pet>)

AnnotationAssertion(rdfs:label :DogLover "Dog Lover"@en)

SubClassOf(:DogLover <http://nomagic.com/ontology/example-case/case-06#Person>)

SubClassOf(

:DogLover ObjectIntersectionOf(

ObjectAllValuesFrom(<http://nomagic.com/ontology/example-case/case-06#has> :Dog)

)

)

)

## Class with Existential Quantification Constraint on Property



Ontology(<http://nomagic.com/ontology/example-case/case-08>

Import(<http://nomagic.com/ontology/example-case/case-06>)

Declaration(

Class(<http://nomagic.com/ontology/example-case/case-06#Person>)

)

Declaration(

Class(<http://nomagic.com/ontology/example-case/case-06#Pet>)

)

Declaration(

Class(:Dog)

)

Declaration(

Class(:DogLover)

)

Declaration(

ObjectProperty(<http://nomagic.com/ontology/example-case/case-06#has>)

)

AnnotationAssertion(rdfs:label :Dog "Dog"@en)

SubClassOf(:Dog <http://nomagic.com/ontology/example-case/case-06#Pet>)

AnnotationAssertion(rdfs:label :DogLover "Dog Lover"@en)

SubClassOf(:DogLover <http://nomagic.com/ontology/example-case/case-06#Person>)

SubClassOf(

:DogLover

ObjectIntersectionOf(

ObjectMinCardinality(1 <http://nomagic.com/ontology/example-case/case-06#has> :Dog) ObjectSomeValuesFrom(<http://nomagic.com/ontology/example-case/case-06#has> :Dog)

)

)

)

## Property Holder with Self-Referential Subproperty



Ontology(<http://nomagic.com/ontology/example-case/case-11>

Declaration(

ObjectProperty(:contains)

)

Declaration(

ObjectProperty(:holds)

)

AnnotationAssertion(rdfs:label :contains "contains"@en)

SubObjectPropertyOf(:contains :holds)

AnnotationAssertion(rdfs:label :holds "holds"@en)

)

## Property Holder with Subproperty



Ontology(<http://nomagic.com/ontology/example-case/case-18>

Declaration(

Class(:Acid)

)

Declaration(

Class(:Liquid)

)

Declaration(

ObjectProperty(:isCorrodedBy)

)

Declaration(

ObjectProperty(:isDissolvedBy)

)

AnnotationAssertion(rdfs:label :Acid "Acid"@en)

SubClassOf(:Acid :Liquid)

AnnotationAssertion(rdfs:label :Liquid "Liquid"@en)

SubClassOf(

owl:Thing

ObjectIntersectionOf(

ObjectMinCardinality(1 :isCorrodedBy :Acid)

)

)

SubClassOf(

owl:Thing

ObjectIntersectionOf(

ObjectMinCardinality(1 :isDissolvedBy :Liquid)

)

)

AnnotationAssertion(rdfs:label :isCorrodedBy "is corroded by"@en)

SubObjectPropertyOf(:isCorrodedBy :isDissolvedBy)

ObjectPropertyRange(:isCorrodedBy :Acid)

AnnotationAssertion(rdfs:label :isDissolvedBy "is dissolved by"@en)

ObjectPropertyRange(:isDissolvedBy :Liquid)

)

## Class with Subproperty without a Range



Ontology(<http://nomagic.com/ontology/example-case/case-16>

Declaration(

Class(:Game)

)

Declaration(

Class(:SoccerMatch)

)

Declaration(

ObjectProperty(:isACompetitionBetween)

)

Declaration(

ObjectProperty(:isPlayedBetween)

)

AnnotationAssertion(rdfs:label :Game "Game"@en)

SubClassOf(

:Game

ObjectIntersectionOf(

ObjectMinCardinality(2 :isPlayedBetween)

)

)

AnnotationAssertion(rdfs:label :SoccerMatch "Soccer Match"@en)

SubClassOf(:SoccerMatch :Game)

SubClassOf(

:SoccerMatch

ObjectIntersectionOf(

ObjectMaxCardinality(2 :isACompetitionBetween) ObjectMinCardinality(2 :isACompetitionBetween)

)

)

AnnotationAssertion(rdfs:label :isACompetitionBetween "is a competition between"@en)

SubObjectPropertyOf(:isACompetitionBetween :isPlayedBetween)

ObjectPropertyDomain(:isACompetitionBetween :SoccerMatch)

AnnotationAssertion(rdfs:label :isPlayedBetween "is played between"@en)

ObjectPropertyDomain(:isPlayedBetween :Game)

)

## Class with Necessary and Sufficient Property



Ontology(<http://nomagic.com/ontology/example-case/case-20>

Declaration(

Class(:CarManufacturer)

)

Declaration(

Class(:Manufacturer)

)

Declaration(

Class(:SteeringWheelManufacturer)

)

Declaration(

Class(:WindshieldManufacturer)

)

Declaration(

ObjectProperty(:hasContractWith)

)

AnnotationAssertion(rdfs:label :CarManufacturer "Car Manufacturer"@en)

EquivalentClasses(

:CarManufacturer

ObjectIntersectionOf(

ObjectMinCardinality(1 :hasContractWith :SteeringWheelManufacturer) ObjectSomeValuesFrom(:hasContractWith :SteeringWheelManufacturer)

)

)

EquivalentClasses(

:CarManufacturer

ObjectIntersectionOf(

ObjectMinCardinality(1 :hasContractWith :WindshieldManufacturer)

ObjectSomeValuesFrom(:hasContractWith :WindshieldManufacturer)

)

)

SubClassOf(:CarManufacturer :Manufacturer)

AnnotationAssertion(rdfs:label :Manufacturer "Manufacturer"@en)

AnnotationAssertion(rdfs:label :SteeringWheelManufacturer "Steering Wheel Manufacturer"@en)

SubClassOf(:SteeringWheelManufacturer :Manufacturer)

AnnotationAssertion(rdfs:label :WindshieldManufacturer "Windshield Manufacturer"@en)

SubClassOf(:WindshieldManufacturer :Manufacturer)

AnnotationAssertion(rdfs:label :hasContractWith "has contract with"@en)

ObjectPropertyDomain(:hasContractWith :Manufacturer)

ObjectPropertyRange(:hasContractWith :Manufacturer)

)

## Class With Property Having Unspecified Multiplicity

UML allows the cardinality of a property to be left unspecified. The concept modeling profile interprets unspecified cardinalities as being zero to many (“0..\*”).



Ontology(<http://nomagic.com/ontology/example-case/case-21>

Declaration(

Class(:SoccerPlayer)

)

Declaration(

Class(:SoccerTeam)

)

Declaration(ObjectProperty(:consistsOf))

AnnotationAssertion(rdfs:label :SoccerPlayer "Soccer Player"@en)

AnnotationAssertion(rdfs:label :SoccerTeam "Soccer Team"@en)

AnnotationAssertion(rdfs:label :consistsOf "consists of"@en)

ObjectPropertyDomain(:consistsOf :SoccerTeam)

ObjectPropertyRange(:consistsOf :SoccerPlayer)

)

# Annex A: UML Conceptual Modeling Profile (normative)

This section defines the UML profile for conceptual modeling and mapping. In order to improve UML’s suitability for modeling real-world concepts, this profile interprets standard with semantic features, as detailed below:

## UML Conceptual Model Primer

A conceptual model can be expressed in UML with the conceptual modeling profile. The profile defines the interpretation of UML concepts used, extends UML concepts with “stereotypes” and makes some UML semantics more specific to conceptual modeling. While there are some extensions, every effort is made to use “generic UML” class diagrams as they are well understood and supported. Readers are referred to the UML specification and the many books and courses on UML for an in-depth treatment of generic UML.

This section is intended as a short primer on how UML is used in this specification to represent conceptual models. The subset of UML used for conceptual modeling is primarily that known as “Class models”, the most commonly used part of UML. Our scope further narrows what we utilize to exclude behaviors and methods – elements used for object oriented design.

Concepts are, of course, the foundation of a conceptual model. Concepts are the elements of how we think about the world. A conceptual model is not a data model, when thinking about concepts we think about the real-world things, not data structures about those things. These real-world concepts become the pivot points around which we define and relate the many data structures that describe those things.

A conceptual model consists of a network of concepts with a simple essential structure. That structure is the definition of classes, relations between them and their characteristics. Classes represent the “things” in our world – including physical things like trees or people and “made up” things like agreements. Other concepts connect those things - the relationships between things-these are UML associations. Things also have characteristics such as weight or color, represented as properties. This basic network of classes, relationships and properties forms the foundation of the conceptual model and define the conceptual framework and vocabulary of a domain. Each of these concepts may be given names, which form the vocabulary of a domain of interest. Various assertions are then made about these concepts and their connections that further define the semantics of those concepts – multiplicities of relationships, specializations between concepts, essential properties of things, etc.

One of the fundamental ways we understand and organize concepts is their arrangement into hierarchies where general concepts are specialized to form more specific concepts within a specific context or with more specific characteristics. A conceptual model can arrange all the fundamental elements into hierarchies using generalization. Another kind of hierarchy is a structural data hierarchy – where data elements contain other data elements. As the conceptual model is not representing data, data hierarchies are not part of a conceptual model – they may be part of data models that are related to a conceptual model. To allow for the many viewpoints that can exist for any concept, a concept can be in many hierarchies at the same time.

The following section defines how basic UML is used to represent the foundational network of concepts using classes, associations and properties. Additional constraints are then attached to this basic framework to enhance semantic expression and the ability of automation to federate and analyze information about those concepts.

#### Classes

Classes classify a set of things according to some set of rules or understanding. Classification is the essential mechanism of conceptualization we use. Classes classify some set of things belonging to that class – this is called the class’s *extent*. Each element of the class is an *instance* of that class – that is it is something the class classifies. Classifications may be arranged in hierarchies.

In the UML conceptual model a class is diagramed as a box with a name at the top. In some cases a definition is also shown next to the box in a “note” form.

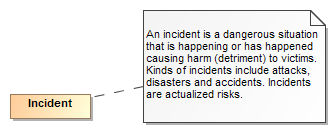


Figure 15 Example of a Class

The above example shows the class “Incident” and its definition. It should be noted that a class is *a way* to classify something. It is natural to classify something multiple ways. For example we may classify a situation as also being a danger or, to someone else, an opportunity to do harm. This is different from many technology models (e.g. Java) that only allow something to be classified in one way and the classification is fixed. *The basic assumption of the conceptual model is that unless specified otherwise, something may be classified in any number of ways and those classifications may change over time*.

#### Instances

While not usually used in the definition of the conceptual model, instances can also be shown in UML and are utilized to illustrate examples. Since the model is conceptual, instances of classes are the “real thing” in the world – not data about them or other technology artifacts. However we sometimes want to show information about instances.

Instances are also shown as a box, but have a “:” separating the name of the instance from its classes.



Figure 16 Instance Example

The above example shows a information about an instance named “Joe Smith” that is classified as a “Person” and a “Victim”.

#### Class Hierarchies

Since Aristotle classes have been arranged in hierarchies – from most general concepts to more specific ones. In UML this is shown as a Generalization – an arrow with a solid line from the more specific concept to the more general. The more specific class is known as the *Superclass* (or *Supertype*) and the more specific the *Subclass* (or *Subtype*). Generalization has some specific semantic rules:

* Everything that is true about the superclass must be true about all its subclasses
* The extent of the subclass is a subset of the extent of the superclass
* All properties and associations that apply to a class also apply to all its subtypes

In a conceptual model a class may have any number of superclasses or subclasses. Some technologies (Like XML Schema) limit the number of superclasses to one.

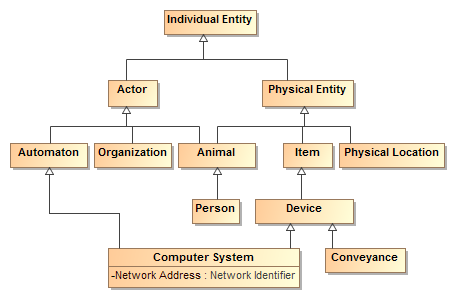


Figure 17 Class Hierarchy Example

The above example shows a class hierarchy with multiple levels.

***Note that all properties and associations defined for all superclasses of a class apply to that class. For that reason a complete understanding of a class and its potential properties must include such superclasses.***

#### Properties

Properties represent qualities inherent in something, such as size, weight or a time. Each property has a “type” for the kind of value that represents that quality. Properties are shown in the box of the most general classifier that can have that quality.



Figure 18 Example of Properties

The above example shows that an animal has the qualities of birthdate, death date, physical sex, height and weight. Note that these is no assumtion that these qualities may be known, required or that different data sources may or may not agree on them – just that a person has these qualties. Instances of properties are facts about the entity they describe.

In conceptual models properties are only used for qualities, never to relate different entities.

#### Associations

Associations describe facts about how entities are related. Associations are shown as lines between the classes that have related instances. At each end of the line is the “association end” – the association end describes how the instances of the class on the far end relate to those of the near end. If there are limits to how many instances may be related, these are also shown. Since an association has two ends, the association may be read in either direction, but is the same “fact”. The ends of relations are typically verbs or verb phrases, but in some cases can also be role names. In either case the name denotes the intent of the class *at the other end of the line*.



Figure 19 Association Example

The above example says that there are relations between actors and activities such that the *actor performs the activity* and the activity is performed by the actor. These are considered two ways to “read” the same fact. Like any fact, relations may be true for some period of time or in some specific situation.

As can be seen in the example the ends of associations are typically verb phrases which can then be read as <the actor> performs <the activity>. In other cases the ends are nouns in which case they represent a role being played. If a role were used above instead of “performed by” it could read: <activity> has performer <actor> (the *has* in this sentence being implied by english gramar).

This combination of classes and associations with ends forms the basis for nouns and verbs common to human language. The terms used for the nouns and verbs should be both consistent with their semantics and resonate with stakeholders – sometimes this is a bit of a challenge.

In some cases the ends of the relation are sufficent to define it, in other cases it makes more sense to give the association a name and its own definition. Associations and association ends, like classes, can be part of a hierarchy.

#### Property and association end hierarchies

Like class hierarchies, property and association end concepts (we will just call both properties from now on) can also be aranged in hierarchies of more or less specific concepts. In UML property hierarchies are represented using with either “Subsets” or “Redefines”. What a property subsets or redefines is shown next to its defintionin in the diagram (Note that by convention this is not shown on summary diagrams, only the primary definition of the property). If a property completely subsumes the other in a particular context it uses a “Redefines” – that is the redefining and redefined properties have the same set of values. If the more general concept can also be used in the context a “Subsets” is used.

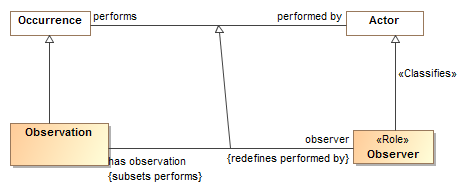


Figure 20 Example of Association End Hierarchy

The above example shows that the “has observation-observer” concepts are specializations of the “performs-performed by” concepts. Observer redefines “performed by” – that is an observation always has an observer, never a “performed by” actor. Likewise “has observation” specializes “performs” but an actor can perform other activities as well. Note the generalization between the associations is impled, but is shown in this example for clarity.

Where a redefined or subset property has no name it is an indication that theproperty is constrained in some way – based on the type of the and or the cardinality (number that may be related), no new properties or associations are actualy defined for a constraint (more on this below).

#### Association Classes

In a conceptual model any “fact” (association or property) may have properties. Of particular importance is the “provenance” of the fact – where the fact came from and thus how much it can be trusted. Facts are also time-bound, true for some period. Every fact is assumed to have these properties. Where an association may have additional specific properties or may it’s self participate in other relationships an “association class” is used. As implied by its name, an association class has both the properties of an association and a class. More complex associations between things use association classes. An association class is diagrammed as an association line and a class box with a dotted line between the association line and its class. While these may seem somewhat visually distinct – they are the “same concept”.

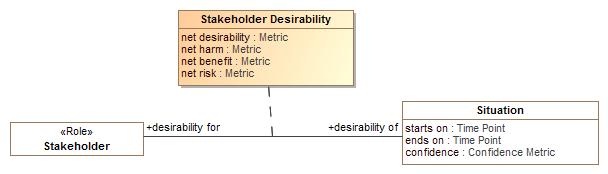


Figure 21 Association Class Example

The above example shows the “Stakeholder Desirability” relation. Between any situation and any stakeholder there can be some metrics as to how much that stakeholder desires or wants to avoid that situation. The Stakeholder Desirability association class represents these as properties of the association: net desirability, net harm, net benefit and net risk – which can all be poitive or negative reflecting a benefit or harm, respectivly.

### Specific kinds of classes

There are additional conceptual modeling specific stereotypes documented in the reference section that further define the semantics of a class. Some of these stareotypes are very important for understanding the conceptual model and are further explained here. These are roles, phases and quantity kinds.

#### Context specific types and <<Classifies>>

Some types may be considered the “fundamental” type of something that is essential to its being for its entire lifetime; this is the default assumption of most classes. Other types classify something in a specific context or for a period of time, SIMF calls these “classifications”. The types an instance is classified with is expected to change over time and may be only valid within a particular context or viewpoint. Classifications are defined with a <<Classifies>> generalization to another type, the type of thing that can be so classified. For example, “Policeman” can classify a “Person”.

Context specific types such as Roles and Phases are classifications and expected to be used in this more contextual and dynamic fashion; these types may be assigned to or removed from an instance over time or in a context.

For an instance to be classified with a classification, it must also have the type of what the classification <<classifies>>. To use the example above, a “Policeman” can’t classify a Toaster since the toaster is not a person. Please see the “Role” and “Phase” discussion for more usage scenarios of <<Classifies>>.

**Implementation note**: most programming languages do not allow for direct representation of multiple classifications, multiple inheritance or context. A common implementation pattern is to represent classifications, roles and phases as independent objects related to the object they classify. An example of this is the IUnkown pattern in .NET.

The following stereotypes define additional classification semantics.

#### Roles

Roles are classes that are expected to be dynamic and contextual, such as teacher, victim or president. A role is defined as a class with the <<Role>> stereotype. Implementation technologies should interpret roles as classifications that may be added to or removed from an instance over time and may be defined in a particular context. A role is usually required to be a role of some particular other class, for example a teacher is expected to be a role of a person (at least until a computer takes her job). The constraint of what a role must be a role of is defined using a <<Classifies>> stereotype of a generalization.

Many implementation languages don’t have the capacity to represent roles, so roles are defined is the single and unchangeable “type” of a class or DBMS table. The problem with this is that the same individual may not be connected across all their roles. Specifically representing roles allows the same individual to play multiple roles and for these roles to change – this better reflects the reality of the world and the way we think about it.



Figure 22 Role Example

The above example shows that an actor can be a person or organization and that either could be classified as being able to play the Owner and/or a Victim role.

Roles help to decouple concepts in models and specifically allow an instance to “play” multiple roles at the same time or over time. Roles, when combined with quantification constraints, clearly define the semantics of roles. For example, we could say that a victim must be a victim of some incident and an owner must own something.

By convention, properties typed by roles may have the same name as the role, this can be read as “has <role>”, e.g. “has victim”, however full verb phrases may be more appropriate in some situations.

#### Phases

Phases are classes that are expected to classify an instance over a specific span of time, such as a teenager, “legal adult” or “Paid Invoice”. A teenager is a person between the ages of 13 and 19 (inclusive) – perhaps “legal adult” is of age 19 or older – we may also want to consider people living or dead, thus alive and dead would be phases. Phase may be considered a synonym for the “State” of something.

A phase is defined as a class with the <<Phase>> stereotype. Like roles, phases use the <<Classifies>> stereotype of a generalization to define what a phase must be a phase of.

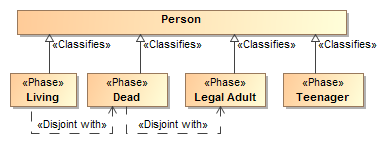


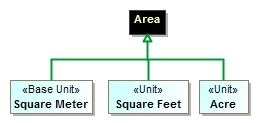
Figure 23 Phases of a person

Also like roles, phases help to decouple concepts in models and specifically allow an instance to “be in” multiple phases (or multiple roles) at the same time or over time. If an instance cannot be in two phases at the same time or be in a role and a phase a “disjoint with” constraint can be used to state that restriction. For example, “Dead” is disjoint with “Legal Adult” and “Living”. Only a “Legal adult” can commit to a contract.

#### Quantity kinds and units

Fundamental to understanding and describing something is physical and other qualities such as temperature, length and color. Many data models fail to capture units of measure explicitly which can and has[[4]](#footnote-4) resulted in dramatic systems failures. A concept for somethings weight should properly be typed by a measure of weight, not an “int” or “real” – which are just ways to represent numbers without knowing what they mean. Of course there needs to be numbers, but in relation to their units.

Figure 24 "Area" Example of quantity kinds and units



In that there are different units that can represent the same kind of measure, such as degrees Celsius and degrees Fahrenheit can represent the same temperature – an abstraction is used above like units. The abstraction for a measurable unit is called a <<Quantity Kind>>. Examples of quantity kinds include Length, mass, temperature, frequency, etc.

As any element of measurement data must be specific to a specific unit in a specific data exchange, the <<Unit>> stereotype is used to define a unit for a quantity kind. A <<Represents>> stereotype of generalization (Diagrammed as a green arrow) is used to say that the unit represents the quantity kind.

In the example above, the “Area” quantity kind (indicated by a black shaded class) can be represented by (the green lines) “Square Meter”, “Square Feet” or an “Acre”. One unit may be nominated as the “Base Unit” and will be used to express conversion factors between the units. As per SI specifications, the Square Meter is the base unit.



Figure 25 - "Animal" example of using quantity kinds.

By convention quantity kinds are used in fully conceptual models whereas units are used in data models. The “Animal” example shows quantity kinds being used to define properties of animals.

### Assertions about concepts

Above we defined the network of essential concepts as classes, relationships and properties. Additional assertions may be made about those concepts using both UML foundational and extended profile capabilities. The following define the kinds of assertions that can be made. Note that the term “property” applies to both simple properties and the ends of associations.

#### Property Ownership

The concept modeling profile of UML interprets the owner of a property *definition* as the subject of that property (its domain) and the context in which that property must conform to certain constraints.

Constraints may be placed on a property. These constraints can include multiplicity, which includes a minimum cardinality and a maximum cardinality, a type for the property, existential quantification, and universal quantification. When an instance is a member of a class, all of that class’ constraints must be met.

Property ownership is not interpreted as “slots” in an object. Property values may or may not be independent of the instance that defined them, thus supporting an OWL/RDF, or “open world”, interpretation of properties and associations.

#### Cardinality

Cardinality defines how many instances of a property may exist for a particular subject instance. For example, how many ages can a person have? The obvious answer is that a person can have at most one age at any one point in time. Thus cardinalities represent the number of instances at any one time.

UML allows the cardinality of a property to be left unspecified. The concept modeling profile interprets unspecified cardinalities as being unconstrained - zero to many (“0..\*”), this is consistent with our general rule that anything unsaid is unconstrained.

#### Global Properties

Global properties are property declarations that can be used by any instance. Normally, a UML property cannot be defined outside of a classifier, so a global property declaration is represented as a UML property owned by a class that is stereotyped as a **«**PropertyHolder**».** The concept of a property holder was introduced in the NIEM-UML standard for a similar purpose. In the concept modeling profile, every property holder is equivalent to one topmost class (⊤) of which all other classes are subclasses. Thus, a property of a property holder is inherited by all subclasses. In addition, while the name of a property holder is irrelevant, consistently naming property holders “Thing”, “Concept”, or “Entity” in all concept models avoids any confusion with normal classes.



Figure 26 Property Holder - "Thing"

### Constraining properties and associations

A cardinality of one or more defined for a property requires that an instance of the related element must exist for an instance of the domain (owning class) of that property or association end to be valid. For example, a living person must have exactly one living brain. This is known as an *existential quantification* (∃) or qualified constraint in first order logic. Existential quantification is defined using UML cardinality and *subsets*.

An existential quantification can be stated for a newly defined property or an existing one. For a newly defined property this is done by simply stating cardinality greater than one. For example, a phone must have at least one button with a “has buttons” association end property and a cardinality of “1..\*”. When a new property is being defined it is given a name. If an existing property is being constrained (without a new property being defined) it subsets or redefines the existing property and does not have a name. In the concept modeling interpretation of UML, any cardinality greater than zero creates an existential quantification constraint.

A property is not limited to a minimum and a maximum cardinality (known as multiplicity) for just one type. A property can have a multiplicity for a superclass, while at the same time having a more specific multiplicity for one or more subclasses of that superclass. This type of constraint is an assertion that, among other possible values, the number of values of one of these subclasses is between some minimum and maximum cardinality.

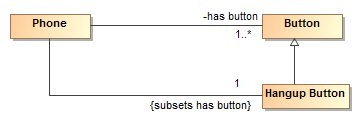


Figure 27 Phone constraint: A phone must have a hangup button

For example, we may say a phone must have any number buttons with a “has buttons” property but exactly one of those buttons must be the “hang up button”. We would then define an unnamed property with the type “hang up button” that subsets the “has button” property with a cardinality of 1. If we wanted the hang up button to also define a new property, we would give that property a name.

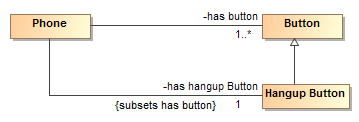


Figure 28 Hangup button with new property

In the concept modeling interpretation of UML, subsetting or redefining a property without giving the new property a different name (or leaving off the new property name altogether) creates a constraint without defining a new property.

As {subsets} or {redefines} with an omitted name is not well defined in UML, in the concept modeling profile it is used to state that a subset of values must meet the stated cardinality and type constraints of the subsetting property. It does not create a new property, although it does create a context in which this constraint holds: the owning class and its subclasses.

The diagram below shows an existential quantification constraint on the global property “is conferred by” (from the property holder “Thing). The multiplicity is such that at least one of the instances of the property constraint must be one of the types in the union.

**Note that the property adding the constraint is unnamed. This is equivalent, in this case, to naming this property the same as the property being constrained (“is conferred by” from the property holder “Thing”).**



Figure 29 Constraining a global property

### Tightening a property’s type

Sometimes it is necessary, in the context of some class, to constrain *all* the values of a property to a particular type. When defining a new property the type of that property assets that all values of that property must be of the given type. This is known as a *universal* quantification or *for-all* constraint (∀) in first order logic. This kind of constraint is an assertion that only values of the specified type are valid, and the number of values must be between some minimum and maximum cardinality.

Where all values of a property must be of a given types in a specialized property, UML *{redefines}* is used. In the concept modeling interpretation of UML, introducing a new property or redefining an existing property creates a universal quantification constraint in the context of the owning class. If the redefined property is given a name, a new property with the quantification is defined. If the redefined property does not have a name the existing property is constrained in the more specialized context (usually a subclass).

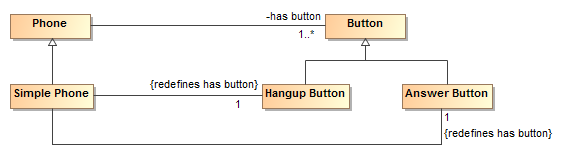


Figure 30 Example of redefines

The example above shows a “simple phone” that has exactly two buttons and they must be an answer button and a hangup button. Since redefines is used, no other buttons are allowed.

The diagram below shows the introduction of a new property “consists of”, defining a universal quantification constraint on the property. The constraint states that, in the context of Soccer Team and any of its subclasses, all values of this property must be of the type “Soccer Player” and that there must be between 5 and 11 values of this property.



Figure 31 Example of cardinality range

The diagram below shows a universal quantification constraint on the property “observer”. Where any occurrence can be performed by any actor, an observation must be performed by an entity in the role of observer.

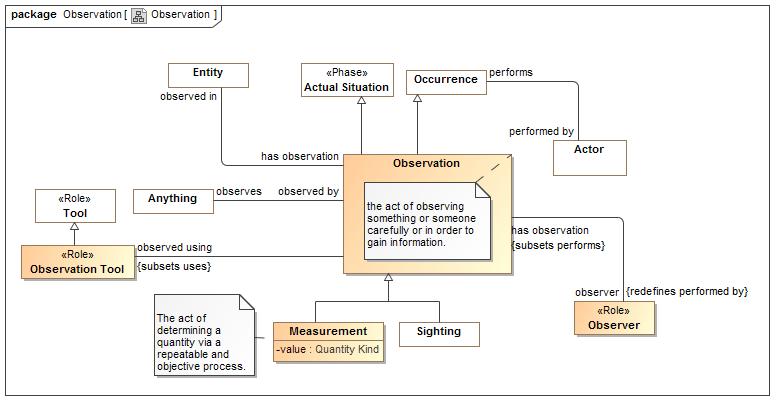


Figure 32 Observation Example

### Inferring a type from its properties

A property's multiplicity or type is declared in the context of an owning class or a property holder. These declarations are always *necessary* conditions for an instance to be a member of the owning class, or, in the case of a property holder, for an instance to be valid at all.

Another kind of condition is known as *necessary and sufficient*. A class with at least one necessary and sufficient condition is known as a *defined* class, which means the differentiating characteristics of the class that make it distinguishable from its parent and sibling classes are defined.



Figure 33 Phone example for sufficient

The diagram above defines a phone as *any “electronic giz” that has a hangup button*. The existence of a hangup button is sufficient to know something is a phone.

In the concept modeling interpretation of UML, a property that has the <<Sufficient>> stereotype applied to it indicates that when an instance satisfies the multiplicity and type constraints for all the <<sufficient>> property’s values, not only is a *necessary* condition for being an instance of the class met, a *sufficient* condition is also met to assume that the domain of that property is of that class. This necessary and sufficient condition allows an inferencing engine to classify that instance as a member of the class that owns the property. All <<sufficient>> constrains must be met for an instance’s type to be inferred.

## Conceptual Modeling UML Profile

The following documents the UML stereotypes defined for Conceptual models, as such not all of these stereotypes are necessarily used within the threat/risk model, but all are available for use. Each stereotype has a mapping to the SIMF Meta model and logical grounding.

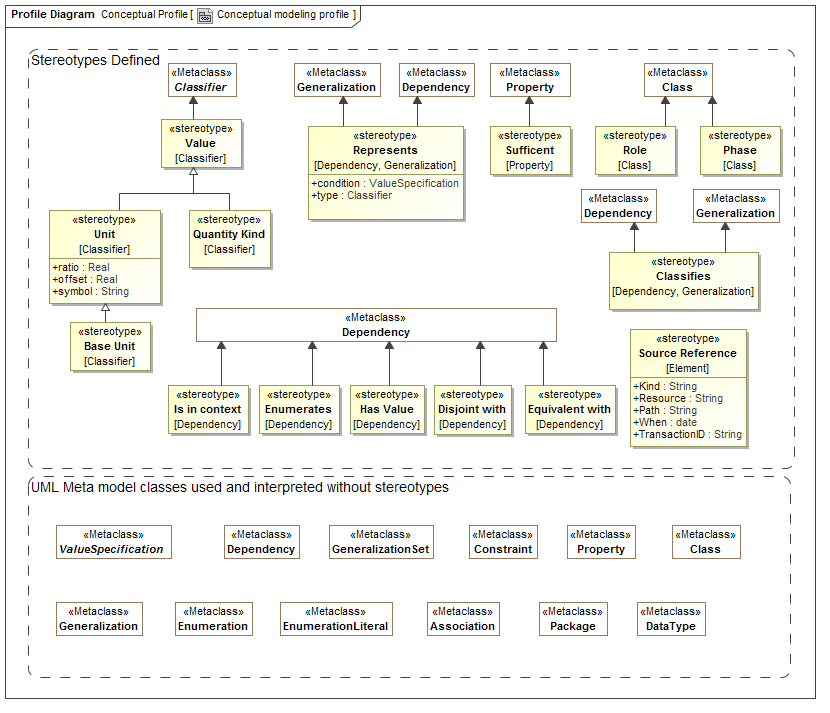


Figure 34 Conceptual Profile

### UML Subset

The SIMF conceptual modeling profile uses a subset of UML, only the following concepts have an interpretation in a SIMF Conceptual Model:

* Association classes (Map to relations - however, all associations and properties are considered “first class”)
* Associations, associations ends {Map to Relation types}
* Cardinalities {Map to Constraint}
* Classes {Map to Entities}
* Data types and primitive types {Map to Values}
* Dependency {Map to in context}
* Enumeration {Map to class and closed set of instances}
* Generalization {Map to generalization}
* Generalization Set {Map to restrictions}
* Packages & Package URI {Map to Lexical and logical context}
* Properties for values {Map to Simple property relations}
* Property defaults {Map to a “Most” quantification}
* Realization {Map to Representation realizes concept}
* Structured classifiers {Map to templates}
* Subsets and redefines of association ends {Map to role generalization}
* Value Specifications {Map to expressions}

SIMF does not preclude other UML constructs; it just does not interpret them. Extensions of the SIMF profiles MAY interpret additional UML elements.

### Stereotype: Classifies [Generalization, Realization]

A classification defined by a <<Classifies>> generalization or realization is an arbitrary classification of an entity beyond any fundamental entity type. <<Classifies>> implies that the subtype is a classification. An instance must be typed by the classified Supertype for it to also be classified as the classifies subtype. A classification may be contextual, such as within a relation, situation and/or timeframe.

Examples

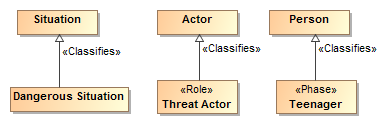


Figure 35 Classifies Example

* A Dangerous Situation classifies a situation.
* Threat Actor is a role that classifies Actor
* A Teenager is a phase that classifies a person..

### Stereotype: Conceptual Model [Package]

A conceptual model represents signs, assertions and classifications for “real world” objects. A conceptual model models the world (or a conceptualization of it), not data or computation. Conceptual models are the foundation of this specification.

Properties

* URI:String (From UML) – the URI of the model

A unique logical identifier for the model.

Examples



Figure 36 Conceptual Model Package

The package “Threat Risk Concepts” is marked as a conceptual model. The URI for this model is: ID://models.modeldriven.org/threat-risk.

Note: As a convention we are using “ID” for “Identity” to distinguish such URIs from internet addresses but otherwise use the HTTP naming convention. The relationship between an ID and any physical resource is outside of this specification.

### Stereotype: Disjoint with [Dependency]

Disjoint specifies that the extent of real-world individuals for which the dependent elements are a sign, classification for or context of have no elements in common.

Note: We do NOT assume that anything has only one type or instance representation unless stated otherwise.

Examples



* No people are also planets
* Fido (A dog) is not Mars (APlanet). {the set of things that are Fido is disjoint with the set of things that are Mars}.
* Fido (A dog) is not a person (Even if Fido’s owner thinks he is). {the set of things that are Fido is disjoint with the set of things that are a Person}.

Disjoint may also be specified using generalization sets.

### Stereotype: Enumerates [Dependency]

Enumerates defines the set of acceptable values for a type, thus “closing the world” for instances of that type. The source of enumerates can either be an instance specification or a package containing instance specifications. The instance specifications must be of the type which is the target of the enumerates dependency. <<Enumerates>> is a more powerful representation of enumerations than a UML Enumeration data type in that <<Enumerates>> instance specifications are not restricted to primitive tag values, however UML enumerations may also be used where appropriate.

The enumerated values are only partially closed in that an additional model may enumerate additional values for a type.

Examples

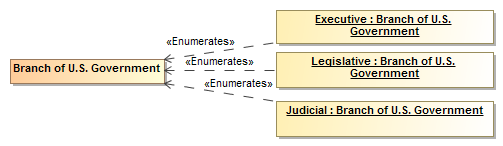


Figure 37 Enumerates Example

### Stereotype: Equivalent with [Dependency]

<<Equivalent with>> is an assertion that two elements in a model or data resource represent the same thing or set of things. Unlike ontological languages it is not assumed that the two elements are consistent as statements from different context may or may not agree.

Examples

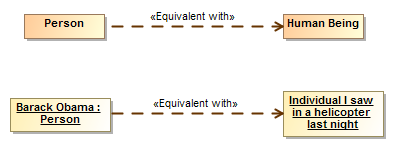


Figure 38 Equivalent with example

* Person and Human are equivelent types, they have the same extent.
* Barack Obama is the person I saw in a helicopter last night.

See also

* ODM/OWL: SameAs, EquivalentClass, EquivalentProperty

### Stereotype: Intersection [Classifier]

An intersection is a class that has an extent which is the complete intersection of the extents of all supertypes. Intersection is a stronger statement than a subtype as a subtype may not be a complete intersection. If an instance has all supertypes the intersection type will be inferred.

For intersection, SIMF considers generalization and realization equivalent. This is due to ownership and legacy considerations in UML. Generalization is the preferred representation.

Examples



All dogs that are pets are pet dogs.

#### See also

* ODM/OWL: intersectionOf

### Stereotype: Is in context [Dependency]

<<Is in context>> is an assertion that the source of the relationship is in the context of the target of the relationship. All assertions and rules defined in the target context apply to the source and everything in the context of the source (in context is transitive). Packages, classes, situations and instances are typical context. Note that <<Is in context>> is the default interpretation of a dependency, if no stereotype is specified it will be interpreted as <<Is in context>>.

#### Examples



* GPS coordinates apply within the context of earth.
* The Radar detector law applies in the context of the state of Virginia and is in the context of rules of the road.

### Stereotype: Phase [Class]

Phase classifies some portion of an entities lifetime (Also known as a “State”).

Examples

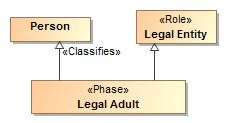


Figure 39 Phase Example

Legal Adult is a phase (of a person) in the role of a legal entity. A Legal Adult represents a phase in the life of a person.

### Stereotype: Quantity Kind [Classifier]

<<Quantity Kind>> is an aspect common to mutually comparable quantities represented by one or more units. Units with a common quantity kind may be algorithmically converted to any other unit of that quantity kind. E.g. temperature. [ JCGM 200:2008].

Quantity kinds are shown as black boxes.

Quantity kinds are the basis for units, typically Units <<Represents>> quantity kinds.

Direct Supertypes

* Value

Examples



Figure 40 Quantity Kind Example

See also:

* JCGM 200:2008
* <<Unit>>

### Stereotype: Represents [Dependency, Generalization]

<<Represents>> is an assertion (rule) that the source type or feature provides a more concrete way to represent the target type or feature. Represents may be used within conceptual models or from a physical model to a conceptual model.

* A representation that is a dependency or realization makes no assumption that the types are substitutable, normally used between a logical/physical model and a conceptual model.
* A representation that is a generalization is substitutable for what it represents, normally used within a conceptual model.

Tag definitions

* Condition: Value specification – expression that must evaluate to true for the represents rule to be valid
* In context: Context – context that scopes the applicability of the representation (may also be specified as a dependency FROM the represents dependency rule).
* type: Classifier – Potentially more specific type that a property is mapped to

Examples

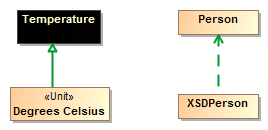


Figure 41 Represents Example

* Degrees Celsius is a way to represent Temperature. Note that there could still be different ways to represent Degrees Celsius (e.g. As a real or integer)
* XSDPersonType (From a physical model) is a way to represent data about a Person (From a conceptual model)

### Stereotype: Role [Class]

A <<Role>> is a classification of an entity based on that entities behavior, participation in a situation or capabilities. A <<Role>> <<Classifies>> the types that may play that role.

Examples



Figure 42 Role Example

Owner and Victim are subtypes of an actor, which could be a person or organization.

See also:

* <<Classifies>>

### Stereotype: Source Reference [Element]

<<Source Reference>> provides traceability to the source of a piece of information in some information resource to facilitate provenance. Source Reference is a statement about the model data and has no semantic implication. Source reference may impact the trust in a statement but the evaluation of trust is outside of this specification. Source is typically populated by the SIMF tooling.

#### Tag definitions

* Kind:String Specifies the kind of resource being referenced. (E.G. “XML”). The values of “Kind” are implementation specific.
* Resource:URL Specifies the location of the resource
* Path:String specifies the location of the information in the source. The form of expression of the path is dependent on the Kind. Kinds and path languages are not defined in this specification.
* When: DateTime Specifies when the information was first asserted in the resource.
* TransactionID:String Transaction ID in which the information was created.

### Stereotype: Sufficient [Property]

The <<Sufficient>> stereotype of a property (or association end) indicates that the existence of the property implies the type of the properties subject. In logic this is known as a *necessary and sufficient* constraint. A class with at least one necessary and sufficient condition is known as a *defined* class, which means the differentiating characteristics of the class that make it distinguishable from its parent and sibling classes are defined.



Figure 43 Phone example for sufficient

The diagram above defines a phone as *any “electronic giz” that has a hangup button*. The existence of a hangup button is sufficient to know something is a phone.

In the concept modeling interpretation of UML, a property that has the <<Sufficient>> stereotype applied to it indicates that when an instance satisfies the multiplicity and type constraints for all the <<sufficient>> property’s values, not only is a *necessary* condition for being an instance of the class met, a *sufficient* condition is also met to assume that the domain of that property is of that class. This necessary and sufficient condition allows an inferencing engine to classify that instance as a member of the class that owns the property.

**All <<sufficient>> constrains must be met for an instance’s type to be inferred.**

### Stereotype: Union [Classifier]

A <<Union>> is any context (Package or Classifier) that has an extent which is the complete union of the extents of all contexts that realize or specialize the Union.

For union, SIMF considers generalization and realization equivalent. This is due to ownership and legacy consideration in UML which do not allow a Supertype to be introduced to a class in another model.

Example



A Marine Mammal is either a Dolphin, Orka or Whale.

See Also

* OWL/ODM: UnionOf

### Stereotype: Unit [Classifier] and Base Unit [Classifier]

Units define real scalar quantities; defined and adopted by convention, with which any other quantity of the same quantity kind can be compared to express the ratio of the two quantities as a number. E.g. Degrees Centigrade, Miles.

Each unit represents a quantity kind using generalization and is thus substitutable for that quantity kind. Typically quantity kinds are used in conceptual models and units in data models.

One unit of a quantity kind may be marked as the base unit. The base unit provides the basis for conversions between units of the same quantity kind. The base unit always has a ratio of one and an offset of zero.

Direct Supertypes

* Value

Tag definitions

* Symbol – the accepted symbol for a unit.
* Ratio – the multiplier by which to multiple the unit to convert to the base unit
* Offset – the difference between zero in the unit and zero in the base unit after the ratio is applied

Examples

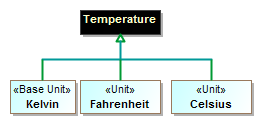


Figure 44 Unit Example

The above defines Kelvin, Fahrenheit and Celsius as units of the quantity kind Temperature. Kelvin is defined as the base unit.

See Also

* Quantity Kind
* Value

### Stereotype: Value

A value is a type representing an atomic unit of information without independent identity. Values include numbers, strings and enumerations. In some cases values may have internal structure. Quantity kinds and units are also values. Values may stereotype any classifier. UML Data types, including primitives and enumerations, are implicitly values.

Examples



Figure 45 Value Example

See Also

* Quantity Kind
* Unit
* Base Unit

### Stereotype: Has value [Dependency]

<<Has value>> identifies an instance specification that contains slots with acceptable values for properties of a type.

Where there is more than <<Has value>> for the same type, if any one <<Has value>> for the same property value is satisfied, the restriction is satisfied.

<<Has value>> is different from enumerates in that enumerates define complete instances that are classified by a type and closes the set of possible instances of a type. <<Has value>> defines property values that must be satisfied by any instance of a type, the set of instances is still open.

Examples

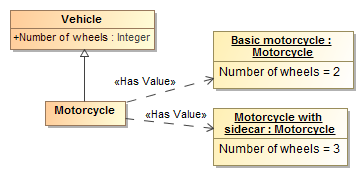


Figure 46 Has Value Example

A motorcycle is a vehicle with 2 or 3 wheels.

See also:

* OWL has value
* Enumerates

## UML Profile – Rules & Data Model Mapping Primer

Rules provide a general framework for stating the consistency of and between model elements. At this time the primary use of consistency rules is for mappings between data models and conceptual models however a <<Rule>> may be used to assert consistency within a model, for example to represent “property chains”.

Mappings define how a particular data model or schema <<*Represents>>* information about the concepts defined in conceptual models. This facilitates an “n-way” mapping of information represented using different data models. Since conceptual models are not data models they do not have any particular representation for “data instances” of that model. Instances of a conceptual model would be the real things in the real world. The real-world concepts are the “pivot points” between the data representations. Of course implementations may automate data models that correspond closely to the conceptual model, but that is outside of this specification.

Due to the various ways to represent information, mappings can become complex. The UML representation of mappings simplifies these mappings and much as possible. Note that details of the mapping relations are defined in the profile specification.

### Representations

The foundation of mapping is the <<Represents>> dependency between classes. Represents says that a particular type found in a data schema model represents information about a concept in a conceptual model.

Example

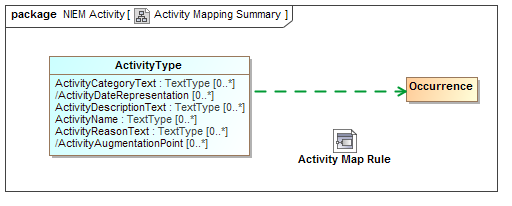


Figure 47 Activity Mapping Summary Example

The above example shows that an “ActivityType” from NIEM-Core represents an Occurrence as defined in the threat/risk conceptual model. By convention we show the represents dependency as a green dashed arrow and do not explicitly show the represents stereotype. Representations provide the highest level of mapping. The diagram also shows that that there is a more detailed activity map rule which will map the properties and relationships between these types.

What this means is that *some* ActivityType instances represent *some* information about occurrences in the “real world”. This also implies that relationships to an occurrence can be validly mapped to relationships of an activity and that properties of an occurrence can validly be mapped to properties of an activity, <<Represents>> relations provide type-safety for mappings.

What this does not say is that ActivityType and Occurrence are equivalent and can necessarily be mapped 1..1. How they are mapped can be detailed in mapping rules. However, if there are no more detailed mapping rules then ActivityType and Occurrence will be mapped 1..1, bidirectionally (mapping of types and properties is considered independent, each property must also be mapped).

### Mapping Representation Rules

The detail of mappings happens in mapping <<Representation Rule>>s. Representation Rules define patterns of data types and patterns of concepts that have map correspondence rules. The map correspondence rules do the real work, mapping element by element.

Mapping representation rules are, externally, not that interesting. They are just a class stereotyped as <<Representation Rule>>.



Figure 48 Representation Rule External Example

The above defines a representation rule for activities that is an assertion that the enclosed pattern must match and provides a context where the map rules are also asserted. If we look inside the Activity Map Rule we see the structure and maps.

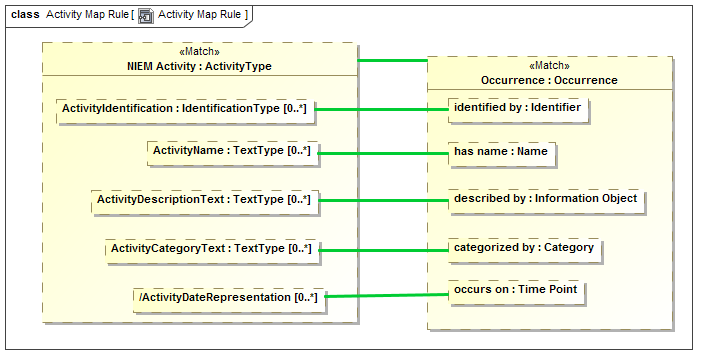


Figure 49 Representation Rule Internal Structure

The above example is the internal “structure” of the Activity Map Rule. In this case the mapping is very 1..1 and simple. Inside of the rule we see “parts” that represent “ActivityType” named “NIEM Activity” and “Occurrence” named “Occurrence”. The green line between them is a “Map” rule, represented as a UML connector stereotyped as “Map”. This states that in this simple pattern NIEM Activities and Occurrences map 1..1. We could also have put filter constraints on that mapping, but in this case did not.

We also see the “<<Match>> on “Occurrence” and NIEM Activity. Match defines the “starting point” for the pattern. A mapping engine will find all instances of Occurrence (in any data format) and map those to NIEM Activity. It will also find all NIEM Activities and map them to Occurrences. All other parts of this mapping become relative to the “Match” elements.

Within both NIEM Activity and Occurrence we see other parts, parts of those types. The green lines create mapping assertions between those parts *within the context of this rule*. This within this rule “ActivityName: maps to “has name”.

A map correspondence is essentially “best efforts”, the types of the mapped elements must either match or have a mapping rule that allows them to be mapped. If, for example, an occurrence had an identifier that was an image and NIEM did not allow for image identifiers, that “fact” would not be mapped.

Mapping for primitive data types, such as strings and numbers, is provided by the mapping engine implementation. This allows, for example, an identifier that is represented as an integer to be mapped to a string.

The important point to remember is that mapping any fact requires that the types are compatible. That type compatibility is defined by represents rules between the types.

### Representation traversals and patterns

The above Activity Map Rule is simple and 1..1, when we get such a simple mapping we shout for joy – because our job is easy. However there is frequently complexity on “both sides” of the mapping – something in the data model may map to multiple things in the conceptual model or require a “Path” through multiple concepts. Likewise there may be intermediate “technical artifacts” that have no real meaning in a conceptual model. This is why we say we are mapping patterns.

For our next example we will look at Incidents, which are a subclass of activities in NIEM and occurrences in the conceptual model. Since these are subclasses on “both sides”, we only need to describe the additional properties of an incident.

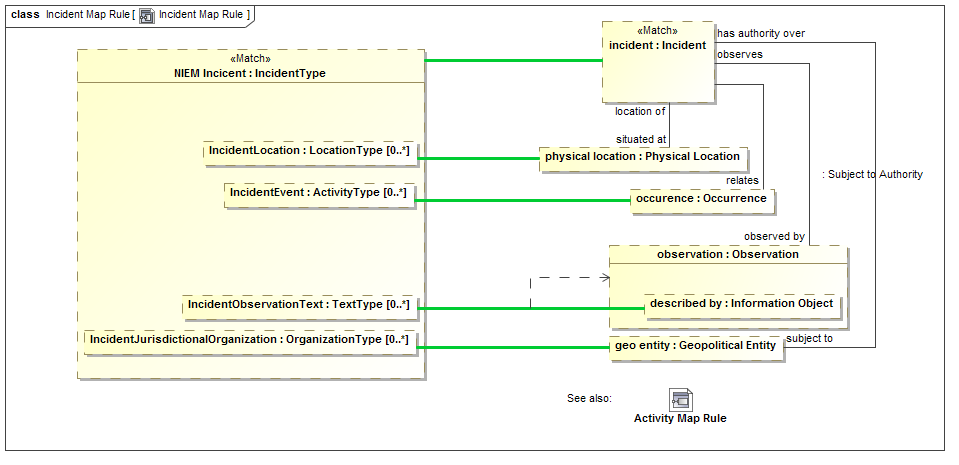


Figure 50 More detailed map rule

The example above shows how a NIEM Incident (named NIEM Incident) maps to a conceptual incident (named “incident”). Incident Map is a subtype of Activity Map so the Activity map rules will all apply to incidents so we don’t need to repeat them here.

We will start with a <<Match>> of “Incident”. Note the line from “incident” to “physical location” labeled “situated at”. The mapping engine will start with an incident and fill in “physical location” iff the “situated at” relationship exists *for that instance* and *what it relates to is a Physical Location*. If that relationship does not exist, “physical location” will be null (empty). Note that physical location could also have multiple values since “situated at” does not have a restricted cardinality.

The values that “end up” in “physical location” will be mapped to “IncidentLocation” in NIEM. Likewise, any mapping in the other direction will hold – any populated “IncidentLocation” will populate “physical location” as well as the relationship to an incident. Once the rule is satisfied, the pattern will hold for all instances of NIEM IncidentType and Incidents.

Now consider the element “described by” within “observation”. This will be populated if the “has observation” relation exists from an incident and that instance has a “described by” property. IncidentObservationText is mapped to “described by” within such an observation. But, in this case, UML notation is a bit misleading, “described by” is a part of the Observation type, not this particular observation part. Since other objects in this rule may have a “described by” property it becomes non-deterministic which “described by” we are talking about. We want to say that we are mapping to the “described by” in the context of the “observation” part. The dependency from the green line to “observation” defines that the context of this map rule is only valid in the context of “observation”, thus making the map deterministic. As many context dependencies as are necessary may be specified for any map rule. All map rules are considered to be in the context of the enclosing Representation Rule. Some tools may report if a map is non deterministic.

### Representation Subsets

Conceptual models use sub classing, multiple inheritance, roles and phases to more accurately and intuitively represent the domain of interest. Many data technologies do not support these concepts and even if they did, would probably structure classes differently. In other cases there may be restrictions on the “extant” of what maps to what that require calculations or other constraints. To provide for these cases we use <<Subsets>> in mapping patterns. A subset defines another part that holds a subset of the instances of the superset part, based on the type and other constraints of the subset part.

To understand this feature we will first look at models for “Entity” and “Actor” in NIEM and the threat conceptual model, respectively.

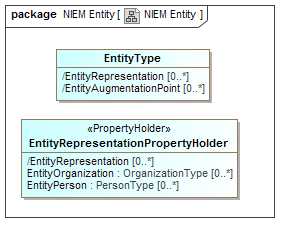


Figure 51 NIEM Entity Example

In NIEM, an “EntityType” has a “substitution group” property with properties that can be “EntityOrganization” or “EntityPerson” to allow the entity to represent one or the other.

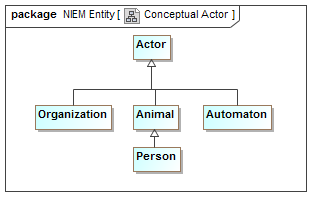


Figure 52 Conceptual Actor Example

In the Threat conceptual model “Actor” is a Supertype of Organization and, indirectly person. It is also a Supertype of “Automaton”. An Automaton can’t be an actor in NIEM so it will not be mapped (However we could define a NIEM extension to allow this).

We want to map actors to NIEM entities, but see that they are very different “shapes”.

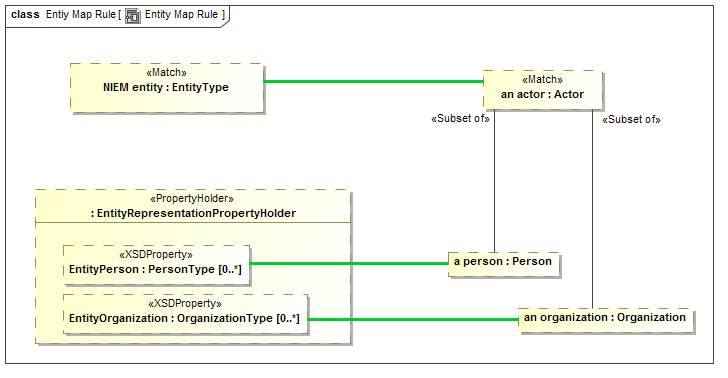


Figure 53 Subset part example

In the above example we see the actor - EntityType mapping. Notice “a person” of type “Person”. “a person” is defined to be a <<Subset of>> actor – that is every actor that is of type “Person” will populate the “a person” part. If an actor is not a Person, “a person” will be null. “a person” is then mapped to “EntityPerson”, a property of “Entity” by way of the substitution group (sorry that this gets into some NIEM substitution group details, but you probably get the basic idea).

Likewise “an organization” will map to EntityOrganization iff “an actor” is an Organization. Note that if “an actor” is neither of these, it will not map to any NIEM entity property.

Note also that there could be other constraints on the subset parts, such as required relations or constraint expressions.

### Map constraints and computations

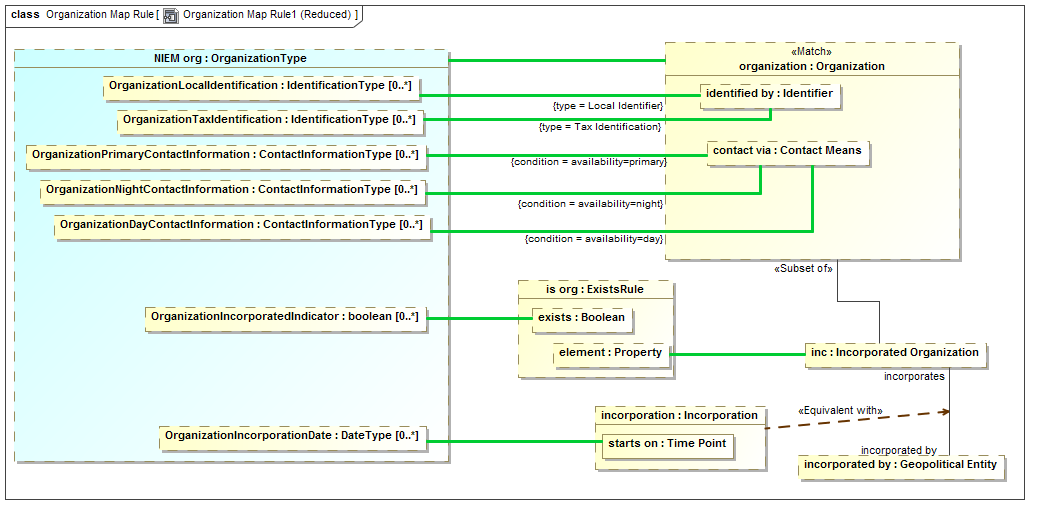


Figure 54 Map constraints example

To continue the tour of the primary mapping capabilities we will look at a subset of the “Organization” mapping.

Note the “type=” on two maps to “identified by”. In the conceptual model there are subtypes of identifiers. In NIEM there are special properties for some of these identifiers. The “type=” constraint on a map says that the map will be constrained to the type (on either end) of the actual instance matched the specified type. So “OrganizationLocalIdentification” will only map to “identified by” if the type if the identifier includes “Local Identifier”. Likewise, “OrganizationTaxIdentification” will only map to “identified by” if the type includes “Tax Identifier” (remembering that a SIMF concept instance can have multiple types). Likewise the reverse is true; those properties will “assert” the type of the identifiers they reference.

On the maps to “contact via” we see “condition=”. The conditions referenced are properties of the association between an organization and “Contact Means”. The maps will be constrained to the “availability” property is set as indicated. Likewise if an organization is being created, that property will be set by the same condition.

Note that “inc” is a subset of an organization only if it plays the role of an “Incorporated Organization”. In NIEM there is a Boolean set if the organization is incorporated. The “ExistsRule” is a computation rule (that is its implementation is outside the specification). But in this case ExistsRule’s behavior is defined – the exists Boolean will be true when the mapped “element” has some value. This results in the NIEM “OrgainizationIncorporatedIndicator” corresponding to the organization being incorporated.

If the organization is incorporated it will have an incorporation relationship to its incorporating body (incorporated by). That incorporation relationship will contain its date of incorporation, which is mapped to the NIEM property. In UML association classes have to be put into a structure like this in two pieces, the “line” and the “box”. Since both the line and the box represent the same “fact”, they are asserted to be equivalent – this is only required when association class properties need to be accessed.

The end result is that the more “flat” representation of an Organization in NIEM is mapped to the conceptual structures.

### Facades and Representation Computations

In some cases it is desirable to have mapping rules as “reusable pieces” that can provide a “Face” to the conceptual model that fits better for one or more mapping rules. There is also the case where these rules fall outside of the expressive power of mapping rules and are best done in calculations (program code).

Facades provide for making a new “face” of either a conceptual model or data model element. A Façade is a class with additional properties and/or relations that can be derived from the element it represents. Either mapping rules or computations are then used to “populate” the façade or map the façade back to what it represents.



Figure 55 Facade Example

The “PersonalInjuryFacade” above represents the concept of “Harm” but only where the harm impacts a Person. In NIEM, injury is only considered relative to a person – so this façade provides such a “View” of the conceptual model, harm restricted to personal injury. In this case no additional representation rule is required, but such a façade could also define new properties or associations that would be populated in the same way as a data model.

Facades can also use “Computations” or Representation Rules do define their properties.



Figure 56 Computation Facade Examples

In the above example both a telephone number façade and address façade are “computed” based on combining both a structured and unstructured representation of telephone numbers and addresses. The specific computation is external to the specification and defined by implementing the two methods of “Representation Rule”: push() and pull(). These methods could be implemented in any language, including “ALF”, the executable language of UML.

When pull() is called on a Façade information will be pulled from the represented objects and populate any other properties or associations. So in the case of phone number, a pull of a structured phone number will populate the unstructured phone number. The reverse is also true.

When push() is called is it assumed the façade is populated from another mapping (in this case a data model) and the result “pushed” into the conceptual model representations – assuming they are defined in the NIEM schema, both the structures and unstructured representations of the same address would be populated.

The mapping engine is responsible for calling push() and pull() at the right times.

In summary, facades and computations provide for reusability and extensibility of mappings.

### Property Chains

Rules may also be used within a conceptual model, an example being the “property chain” concept from OWL which allows a “path” through properties to be summarized by another property.



Figure 57 Property Chain Example

In this example we see a simple model of a person with parents and male people that can be prothers or uncles. The “Uncle Rule” states that the “path” through “has parent” to “has brother” <<Map>>s to “has uncle”.

## Consistency Rules Profile

This section specifies the stereotypes specific to rules and data model mapping.

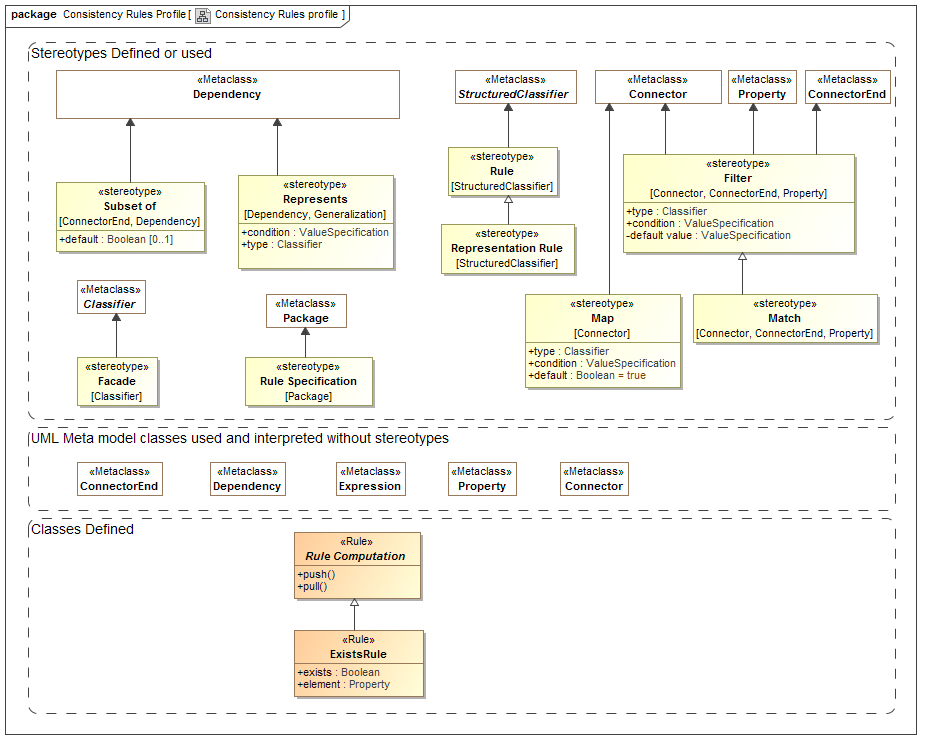


Figure 58 Rules Profile

### Class: ExistsRule

ExistsRule is a rule to map the existence of an element to a Boolean. exists is true if element is not null. ExistsRule is used within Representation Rules.

Properties

* Exists : Boolean – true when the part mapped to element in not null.
* Element : UML::Property – reference to a property for which the NULL value corresponds to the truth value of Exists.

Example

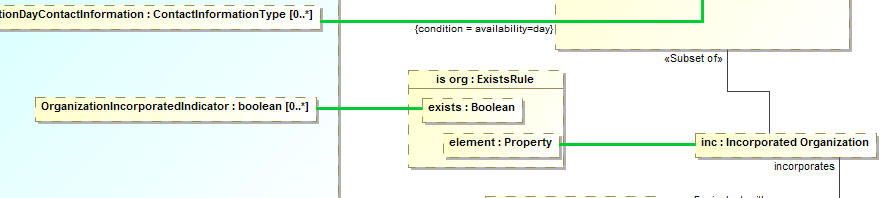


Figure 59 Exists Example

The above will make the Boolean “OrganizationIncorporationRule” correspond with the organization being an Incorporated Organization”.

### Stereotype: Facade [Classifier]

Façade classifies an element as being a view of (facade of) one or more other elements. Facades usually define additional properties that match some external view of a conceptual model element and then ”fill in” those elements on one of several ways.

A Façade should use a generalization with a <<Represents>> stereotype to specify the class for which it is a façade.

Note that a façade may be used as a pure specification or may actually be instantiated, depending on implementation styles.

Façade’s will use one of three methods to relate the façade properties to the conceptual Model:

* A <<Representation Rule>> - where the representation rule defines mappings
* Subclassing “Representation Computation” – where the computation methods implement the façade.
* Constraints on the definition of the façade to subset the represented concept.

These methods may be combined.

Examples



Figure 60 Facade constraint example

The above “PersonInjuryFacade” of “Harm” will be a subset of Harm such that the harm is to a person.



Figure 61 Computation Facade Examples

The above “Postal Address Façade” and “Telephone Number Façade” will be populated by the execution of the “push() and pull() methods as defined in “Representation Computation”. In these cases the façade will convert between a textual and structured representation of phone numbers and addresses.

### Stereotype: Filter [Property, Connector, ConnectorEnd]

<<Filter>> categorizes a connector or property within a rule as having a condition (A Boolean expression restricting the population of elements) and/or a default (an expression creating a new object where one does not exist). The expression syntax may be ALF or OCL. Filter may also provide a default value where the filtered element is null.

Filter may be used on the end of a <<Map>> to qualify the map.

Tag Definitions

* Type – a required type of the element
* Condition – an expression (ALF or OCL) returning a Boolean. If the condition is true the filtered element will hold.
* Default – an expression returning an instance to populate the element if that element is NULL.

*Please see Map Example.*

### Stereotype: Map [Connector]

<<Map>> defines a rule that maps the representation of elements between levels of abstraction, data representations or property indirections. A Map connector asserts that the values on both sides will be equivalent. A map is only asserted within the context of the enclosing rule where the “Match” pattern is valid.

A <<Map>> will also assert a conversion between types if any conversion rule exists between those types, so a map from a string property to an integer property will attempt a string/integer conversion.

**Map constraints**: The ends of a map (ConnectorEnd) may have a <<Filter>> defined to further constrain a map or the same properties may be set for the <<Map>> as a whole, in which case it applies to the conceptual model. If the filter type is set, the end must match the more constrained type specified. If the filter condition is set, the condition must hold for the map to be applied.  
**Map dependencies**: A map may also have dependencies to parts, the map must be in context of these parts. The mapping will be to the parts within the dependent parts. Dependencies are only required where the map would otherwise be indeterminate. The default context for maps to nested parts is the <<Match>> element. Map dependencies are usually required due to the non-obvious notation of UML where parts within parts simply showing parts of the type, not parts of the part.

Tag Definitions

* Type – a required type of the element in the conceptual model
* Condition – an expression (ALF or OCL) returning a Boolean. If the condition is true in the conceptual model, the map will hold.

Examples

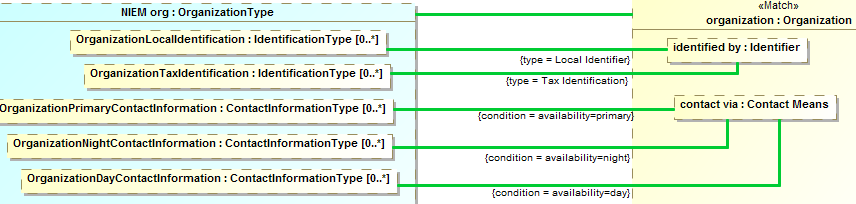


Figure 62 Map examples

The above shows multiple “Map” connectors. Those to “identified by” will be constrained to instances where the identifier is of the indicated type. The “condition” maps will only hold to where the expressions are true or will assert the condition to be true.

### Stereotype: Rule Specification [Package]

A <<Rule Specification>> package represents rules for model consistency and mapping data between different models. Defining a package as a rule specification asserts the enclosed rules.

Properties

* URI:String (From UML) – the URI of the model

Examples



Figure 63 Rule Specification Example

### Stereotype: Match [Connector, Property]

Match specifies an element or set of related elements in a representation rule structure that must match the model, these are the predicates of the rule. The <<Match>> elements form the foundation of the pattern to match and the root of any traversals. Match is s subtype of “Filter” and can utilize the condition and default tags. Match is bidirectional, the matching process may start at either “end” of the mapping.

Supertypes

* Filter

Example

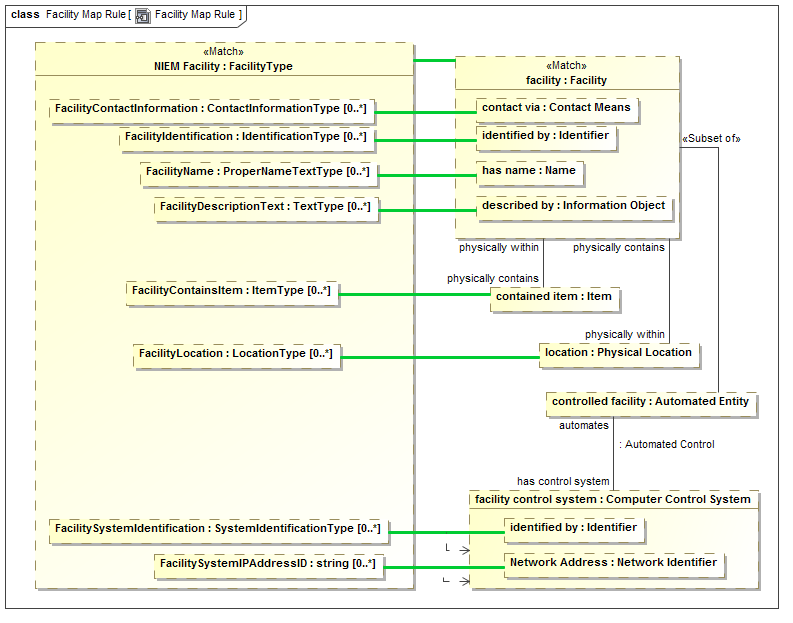


Figure 64 Match Example

In the above example “facility : Facility” is specified as the <<Match>> part on the concept6ual model side. On projecting from the conceptual model all Facilities will match this element (since it has no other constraints). On production from “NIEM facility:FacilityType” the mapped FacilityType will be the basis for the match.

The facility will then be the root to traverse across the “physically contains”, “physically within” and “has control system” relationships.

### Class: Rule

A rule defines something which must hold true for the context of the rule.

A rule is a pattern structure described by a structured classifier that shows how elements are related. Each mapped must match, including any traversals through structures defined with properties and connectors. Such traversals are links which may also have filters to more precisely define the pattern. The mapping engine ensures that the patterns match, bidirectionally.



Figure 65 Rule Example

The above defines an “Uncle Rule” that asserts that “has uncle” will have the same value as the “path” through “has parent” and “has brother”.

See also: OWL “property Chain”

### Class: Rule Computation

A rule computation is a class or facade that includes calculations as defined by push() and pull() methods. Rule computation allows for programmed computations. The implementation may be external or utilize ALF, the UML execution language. A Rule computation should <<Represents>> a class that will be the basis for computation. Facades that are to be calculated should subclass Rule Computation.

Operations

* Operation: push() - An operation called to evoke the behavior associated with a new facade element being created or modified.
* Operation: pull() - An operation called to evoke the behavior associated with a facade representing existing elements.

Please see example under “façade”

### Stereotype: Representation Rule [Structured Classifier]

A representation rule is a pattern structure described by a structured classifier that shows how both "sides" of a representation are related. Each "side" must match, including any traversals through structures defined with properties and connectors. Such traversals are links which may also have filters to more precisely define the pattern. The mapping engine must ensure that the patterns match, bidirectionally. Representation Rules may contain and contextualize other rules, in particular <<Map>> connectors are used within a representation rule to map between parts of the representation rule.



Figure 66 Representation rule definition

A representation rule is just a class stereotyped as <<Representation Rule>>. The interesting part is the internal structure.

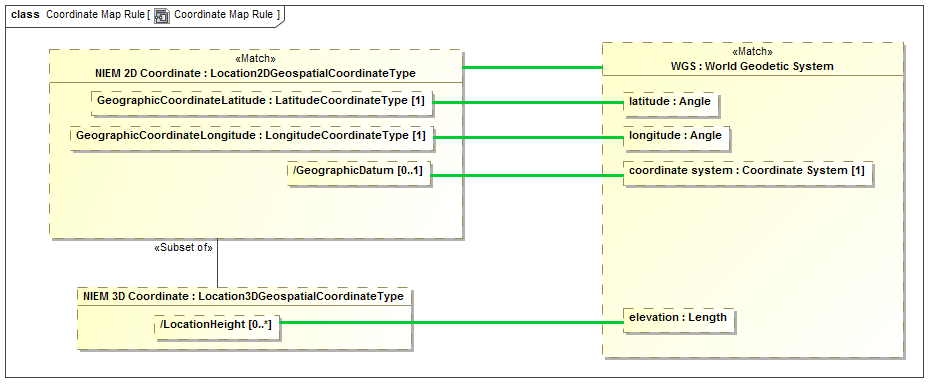


Figure 67 Structure of example representation rule

The internals of a representation rule, shown as a UML composite structure diagram, contain parts and connectors that use the other stereotypes defined in this section. The “root” of the rule is marked with <<Match>>. The detailed work of mapping properties is done with <<Map>> rules inside of a representation rule.

### Stereotype: Represents [Realization, Dependency]

Please see the definition of <<Represents>>in the conceptual profile. This entry discusses use of <<Represents>> for mapping.

When used within a mapping specification the dependency form of <<Represents>> is used between a data model and a conceptual model. It states that the data model element is a data representation of the concept. <<Represents>> is usually the starting point for a mapping, where the “high level” types are connected.

Represented types are used to filter the relations between instances to only those relations that are valid for the represented types. For example, if any entity can be “related to” any other, and that relation is mapped to a property “relatedToIncident, with type incident, only the actual relations to an incident will be mapped – thus providing for mapping type safety.

If not other representation rules are defined for represented classes, the classes are defined to be mapped 1..1.

Tag Definitions

* Condition – a Boolean expression that must be true for the represents rule to be enforced.
* In context – other elements that must be in context for the rule to be enforced. In context may also be specified by an un-stereotyped dependency.
* Type – a constraint on the type of the represented element.

Examples

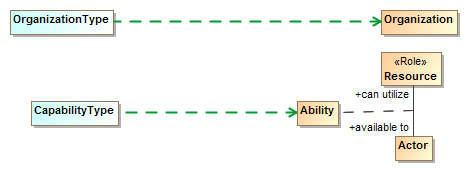


Figure 68 Represents dependencies

The above states that OrganizationType is a data representation of an organization and that CapabilityType is a data representation of an ability ( an association class between a resource and an actor with that capability). Represented types are used to filter the relations between instances to only those relations that are valid for the represented types.

### Stereotype: Subset of [Connector End]

In a pattern or mapping rule, <<Subset of>> defines a part (UML Property) that represents a subset of another part. The subset may be constrained by a more specific type, expressions or required cardinalities. The <<Subset of>> stereotype is applied to the end of a connector that is the superset of the subset part.

A part within a structured classifier may hold any number of instances, including zero. The superset property is assumed to hold some value for which the subset property is a constraint, forming the subset. The subset may be constrained by its type (usually a subtype or role of the superset) or by required associations or filter conditions.

Tag Definitions

* Default:Boolean – indicates that the subset be a subset that excludes all other subsets of the same part.

Examples

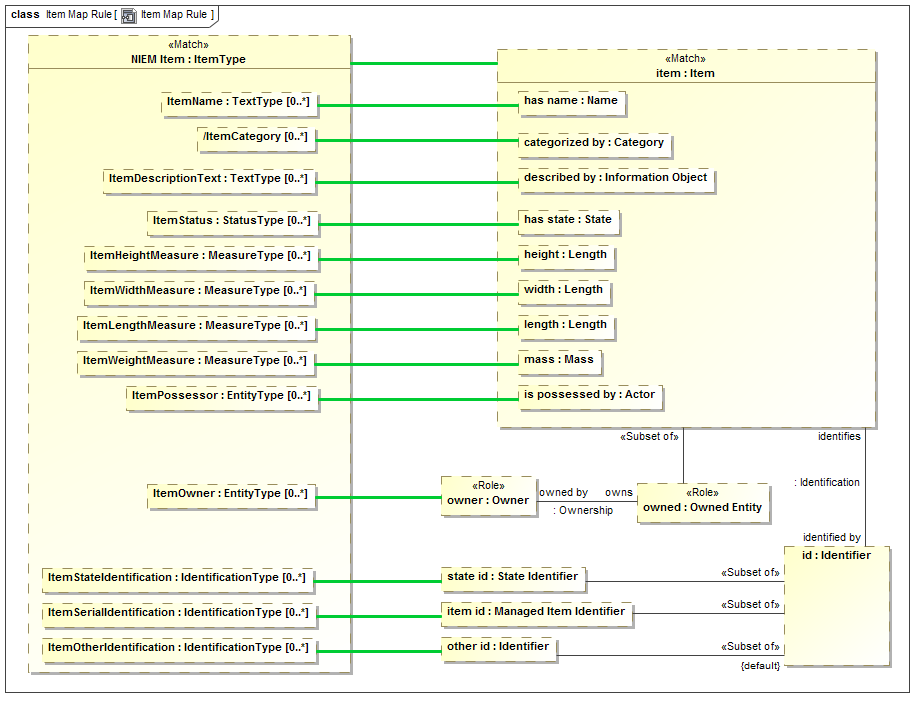


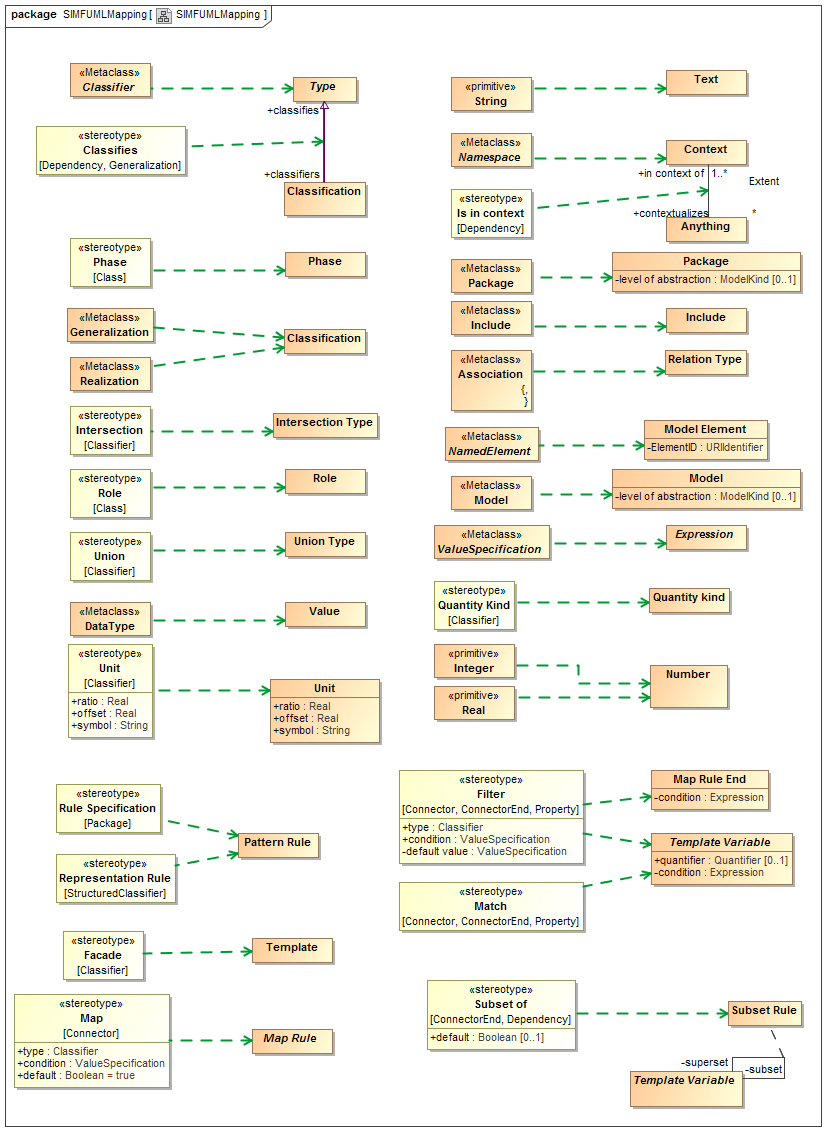
Figure 69 Subsets Example

The above uses subsets for the following:

* “owns :Owned Entity” – is populated when item has a role of Owned Entity
* “state id” is a subset of “id” when the id has the type “State Identifier”.
* “item id” is a subset of “id” when “id” has type “Managed Item Identifier”
* “other id” is a subset of “id” less the subsets “state id” and “item id”.

# Profile mapping to metamodel (Normative)

Note that this is a preliminary profile/meta model mapping.



# Concept Index

Actuality, 44

Adjective, 53

Adverb, 53

All, 50

Always, 54

Antonym, 54

Anything, 56

applies to, 43

asserted in, 46

asserts, 57

at once, 41

Base Unit, 36

Binding, 44

binds, 46

Block, 33

bound in, 45

bound in context, 30

Bound Role, 30

Bridging, 35

Broader than, 54

Calculation type, 30

calls, 30, 31

categorizes, 62, 63

changed by, 59

changed in, 34

changes, 60

Classification, 61

classifiers, 63

classifies, 61

Classifies, 93

Computed Fact, 31

Concept, 56

Conceptual, 35

Conceptual Model, 93

condition, 41, 42, 49

Conditional Rule, 41

Constant Reference, 31

Constrained Relation Type, 39

constraining type, 42

Constrant, 41

Context, 57

contextualizes, 57, 58

created by, 56

creates, 60

defined by, 56, 57

defined in, 56

defines, 34

Definition, 57

describes, 57

Disjoint with, 94

Documentation, 57

ElementID, 34

Enumerates, 94

Equivalent with, 95

Exactly One, 50

ExistsRule, 110

Expression, 31

Extent, 57

External Transaction, 59

Facade, 111

Filter, 112

from, 31, 42

from type, 42

Function Call, 31

function result, 32

Function Type, 31

Future, 54

General Constrant, 42

Generalization, 58

generalizes, 57, 58

Group, 53

Has constituent, 54

has key, 63

has rule, 57

has type, 56, 62

Has value, 101

Identification, 51

identified by, 51, 57

Identifier, 51

identifies, 51, 52

identifies instances of, 63

identifying role, 63

impled by, 60

in context, 60

in context of, 56, 58

in situation, 45

Include, 34

Individual, 58

Individual Type, 62

Inferred Transaction, 59

Information Type, 62

initial situation, 46

Instance Relation, 62

Intersection, 95

Intersection Type, 62

invalidated by, 56

invalidates, 60

involves, 46

involves from, 40

involves to, 40

Is in context, 96

is used by, 30, 32

isPrimary, 63

isUnique, 62

Key, 62

label, 52

level of abstraction, 34, 35

Lexical Reference, 34

Lexical Scope, 34

Lifetime, 54

magnitude, 37

Map, 112

Map Rule, 42

Map Rule End, 42

Match, 114

maximum number, 42

mininum number, 42

Model, 34

Model Element, 34

ModelKind, 35

Most, 50

Multiplicity Constraint, 42

Name, 52

name text, 52

Name Text, 52

Narrower Than, 54

No, 50

Noun, 53

number, 52

Number, 36

Object Management Group, Inc. (OMG), x

Object Operation Type, 32

Occurance, 45

offset, 38

OMG specifications, x

Package, 35

part, 52

Part of Speech, 53

Past, 54

Pattern Rule, 43

Perspective, 35

Phase, 45, 63, 96

Physical, 35

Plural, 53

precision, 37

prefered term, 57

Present, 54

Property Chains, 108

quantifier, 49

Quantifier, 49

quantifies, 49

Quantity, 37

Quantity kind, 37

Quantity Kind, 96

ratio, 38

reciever, 32

redefines, 45

Redefinition, 45

Referenced scope, 34

references, 30, 35

related term, 53

Related word form, 54

relates from, 46

relates to, 46

Relation, 40

relation kind, 53

Relation Type, 40

Representation Rule, 115

Represention, 35

Represents, 97, 116

Return type, 32

Role, 45, 98

Rule, 43, 114

Rule Computation, 115

Rule Specification, 113

Rules, 35

Singular, 53

Situation, 46

Situation Type, 46

Source Reference, 98

specializes, 57, 58

state of, 63

sub transaction, 60

subject term, 53

subset, 48, 49

Subset of, 117

Subset Rule, 48

Sufficient, 98

super transaction, 60

Supersedes, 54

superset, 48, 49

symbol, 38

Synonym, 54

Template, 48

Template Rules, 49

Template Variable, 49

tense, 52

Term, 52

Term Number, 53

Term Relation, 52

Term Relation Kind, 53

term relations, 52

Term Tense, 54

Text, 37

TextDefinition, 57

There Exists, 50

timestamp, 59

to, 32, 42

to type, 43

Transaction, 60

transacts, 58

Traversal, 32

traverses, 32

Type, 63

typographical conventions, xi, 17

Union, 99

Union Type, 63

Uniqueness Constrant, 43

Unit, 37, 99

uri, 53

URIIdentifier, 53

value, 31

Value, 38, 100

Verb, 53

1. ECSS is an initiative established to develop a coherent, single set of user-friendly standards for use in all European space activities [ECSS]. [↑](#footnote-ref-1)
2. Not to be confused with how “domain” is used in the intelligence community, which refers to levels of security [↑](#footnote-ref-2)
3. “transaction” in this context simply means a set of statemetns asserted at once, it may or may not have ACID properties as do DBMS transactions. [↑](#footnote-ref-3)
4. https://en.wikipedia.org/wiki/Mars\_Climate\_Orbiter [↑](#footnote-ref-4)